DEPUSH HexCrawler: Mechanical and Control System Improvement

A Major Qualifying Project
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfilment of the requirements for the
Degree of Bachelor of Science by:

YunSoo Choi (ME) Arthur Dutra IV (RBE) Daniel Earley (RBE)

In Collaboration with Huazhong University of Science and Technology
Partners: Shen Yuwen, Song Lu, Xu Rui, and Zhang Di

Date: 24 August 2010

Approved:

Professor Yiming (Kevin) Rong, Major Advisor, ME

Acknowledgements

The team would like to thank Huazhong University of Science and Technology for their support and hospitality during the entirety of our project. We would not have been successful without assistance from their faculty and staff, and the resources they provided to us. We would also like to thank our advisors, Professor He Lingsong from HUST, and Professor Kevin Rong from WPI, for their guidance prior to, during, and after the project. Lastly, we would like to thank our partners from HUST. Together, we overcame cultural boundaries, learned from each other, and surprised many with what we were able to accomplish in such a short time.

Abstract

The DEPUSH Company purchased the rights to sell the HexCrawler educational robot. An open-loop control system and a simple walking mechanism made the robot unstable and incapable of performing to current Hexapod standards. DEPUSH asked our team to update the HexCrawler so they could offer a competitive educational robotic platform to their customers. We approached the project from two fronts. Our first goal was to redesign the robot's chassis and legs to increase mobility and range of motion. Our second goal was to implement a more powerful control system that could precisely manipulate the robot's limbs and leave enough overhead to add more functionality at a later date. The resulting product was a cutting-edge hexapod capable of 6 Degree-of-Freedom body control and a prototype high-level computer interface. As it stands now, HexCrawler 2.0 is a versatile platform with potential applications in a variety of robotics-related projects and solid foundation for future research on high-level control.

Table of Contents

Table of Figures

List of Tables

Chapter 1: Introduction

1.1 General Information

The expectations of the project from DEPUSH were to improve the functionality of their HexCrawler robot in order to better serve a college level educational market. In order to achieve these goals, improvements to the capability of both the mechanical and control system on the HexCrawler robot would be required.

1.2 Specific Problem

The HexCrawler robot is a six-legged walking robot roughly 50cm long by 40 centimeters wide, with all components fabricated from 5052 alloy, 1/16” aluminum. Each leg of the robot has two degrees of freedom (DOF) provided by two HS-322HD servo motors, thus yielding a total of twelve servos on the robot. The main embedded controller on the robot is a Parallax BASIC Stamp 2 (BS2), with a single Propeller Servo Controller (PSC) daughter board used to control the PWM signal generation for the twelve servomotors.

To improve the mobility by allowing the robot to better adapt to various terrain obstacles with a smoother walking gait, the size of a leg’s workspace environment would have to be increased to allow for a wider range of motion. With only two degrees of freedom, the leg workspace is confined to two-dimensional, infinitely flat areas, making accurate straight-line operation in a three dimensional workspace all but impossible. An additional degree of freedom added to the leg mechanism (via an additional servo joint) would rectify this issue.

Due to limitations of the BASIC Stamp 2 embedded controller, the calculations necessary for the inverse kinematics of the six legs would surpass its capabilities. In order to meet and exceed these requirements (to allow for future expandability), the chosen Cortex-M3/STM32 embedded controller had the best compromise of features, functionality, ease of use, and availability in the secondary educational market. The improved controller must be able to handle improved functionality such as inverse kinematic calculations, environment mapping, path planning, and distributed computing.

1.3 Objectives

Improvement of Mobility – Improve the mobility of the HexCrawler educational robot by creating a full six-degree of freedom body.

Improvement of Stability – Reduce the jerk of the robot body while in motion by implementing advanced kinematic algorithms and closed-loop feedback.

Terrain Adaption – Addition of closed-loop feedback by using ground detection sensors to adjust the inverse kinematics algorithms.

Stair Climbing – Provide a challenge for the secondary educational robotics market by allowing the robot to be capable of climbing stairs.

Economical Price – Keeping the cost of the improved prototype HexCrawler 2.0 robot under 7,000 RMB would be 30% lower than the retail price of the current HexCrawler.

1.4 Challenges

Walking Mechanism – The addition of the third degree of freedom will add additional mass and volume to the robot mechanism, thus elevating the necessary torque requirements and making efficient packaging of the components more difficult. To increase the opportunities for closed-loop feedback, mounting slots for additional sensors will have to be provided. Keeping the complexity and number of discrete parts to a minimum and the number of standardized parts to a maximum are desirable for reducing manufacturing costs.

Cortex-M3/STM32 Embedded Controller – The substitution of the BS2 for a more powerful, C-programming based embedded controller would necessitate either writing all new low-level device firmware and high-level control software, porting existing open-source software libraries to the specific Cortex-M3/STM32 controller and development board, or some combination thereof.

1.5 Expected Results

Pending successful completion of the project, the improved HexCrawler 2.0 prototype would encompass improved mechanical features and control functionality over the existing HexCrawler robot. This would make the robot better suited for the secondary educational robotics market.

Chapter 2: Background

2.1 DEPUSH Company Profile

2.1.1 Company History

DEPUSH Technology Co, Ltd. was founded in February 2001 in the Futian district of Shenzhen, PR China to create and sell educational and innovative laboratory solutions for the advanced engineering educational market. Since its inception, the company has expanded to two branch offices in Wuhan, PRC and Beijing, PR China respectively, maintained strong, long-term strategic cooperation with Huazhong University of Science and Technology in Wuhan, PR China and Shanghai Jiaotong University in Shanghai, PR China and works closely with the Chinese Academic Automation Institution, Shenzhen Polytechnic Institute and Parallax Inc., a US-based educational robotics company.

DEPUSH provides a variety of innovative products designed for the educational engineering laboratory environment, including but not limited to laboratory equipment and construction, consultation, system development and teacher training for Advanced Institution Vocational Technical College. Their full range of products and services are specifically designed for mechanical engineering, electrical and electronic engineering, information engineering, material science and controls engineering, but can also be used for generic engineering education.

2.1.2 DEPUSH Products

DEPUSH currently offers products for facilities in engineering education in two main product lines: the DRLab and DRRob series. For the prior, DEPUSH offers “DRVI Reconfigurable Virtual Instrument” and “DRLink fast reconfigurable completer controlled platform” as the core for a variety of electronics, object detection, and control technology courses via three creative, experimental laboratory exercises. These three laboratory exercises (“Electrical and Electronic Innovation Lab”, “Detection and Sensor Innovation Lab,” and “Control Theory and Control Engineering Innovation Lab”) include the experimental platform hardware and software.

The DRRob product line are robotics platforms designed to educate students about machinery, electronics, sensing, and control theory of an advanced experimental platform, as well as provide a base platform suitable for basic training/laboratory exercises, advanced laboratory exercises, or any open research in mobile robotics applications. The main products of this product line are their Basic Educational Robot, Professional Robots (including industrial and intelligent mobile-ground robots), and Research Robots (with the capability for complex robot subsystems).

2.1.3 Competing Companies and Products

Beijing Torch S&T Co, Ltd

Founded in Beijing, PR China in 2005, Beijing Torch S&T Co, Ltd. is a professional electronic equipment manufacturer and service provider, specializing in the development, production and sales of surface mount (SMT) printed circuit boards and research and development supplies. They currently market a much simpler six-legged walking robot for 100 RMB.

Figure 1 – Beijing Torch S&T Co. Ltd. Hexapod

Phoenix Technologies Ltd

Founded in Beaverton, United States in 1979, Phoenix Technologies has been at the forefront of the development of innovative educational products for the engineering market. From the beginning, their products emphasized the importance of systems engineering approach using embedded controllers with physical mechanisms. One of their most recent products is the Phoenix Hexapod robot, a small six-legged walking robot with three degrees of freedom per leg with a current market price of $1578.42 USD.

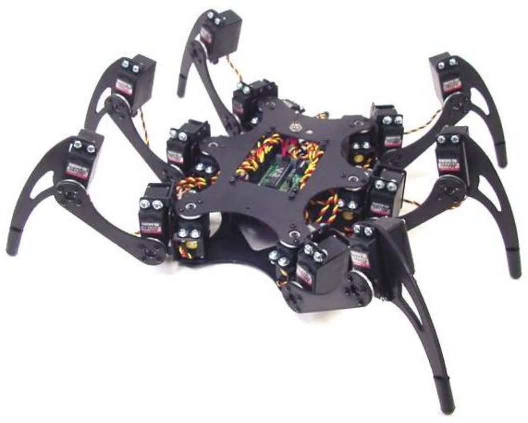


Figure 2 – Phoenix Hexapod

2.2 Company Motivation

The rights to the HexCrawler robot were purchased by DEPUSH from a (now discontinued) model from a US-based company prior to the project. Since DEPUSH had not yet made any modifications to the design, and since the design itself was almost a decade old, it was due for a redesign. While modern embedded controls powerful enough to run full inverse kinematics and environment mapping algorithms are relatively inexpensive, such processors would have been near the top of the line a decade ago, making them prohibitively inexpensive. Thus, the limited processing power available in embedded controllers at the time necessitated simpler mechanical designs for financial reasons.

Due to advances in technology and the secondary robotics education market, DEPUSH was looking for a new product to adapt to these changing conditions and meet current expectations from an educational six-legged walking robot. To keep up with short-term future advancements in robotics technology, additional features were specified that should be taken into consideration in the planning and design process. While these features would not be used in the scope of this project, they would accelerate future development and would require fewer updates to the HexCrawler product line to stay competitive.

2.3 Details of the Current HexCrawler Design

2.3.1 Mechanical System

The HexCrawler robot was fabricated using anodized 5052 alloy, 1/16”-thick aluminum sheet metal. It is 49.68 x 40 cm in size and has a height that ranges between a maximum of 15.24 cm to a minimum of 12.34 cm. The robot has a mass of 1.81 kg and a lifting capacity of 3.4 kg. The walking mechanism was designed using legs with two degrees of freedom, with one of those degrees of freedom provided by a four-bar linkage. Mounting slots on the lower, upper, front, and rear sheet metal panels allow for future expandability with additional sensors and components.

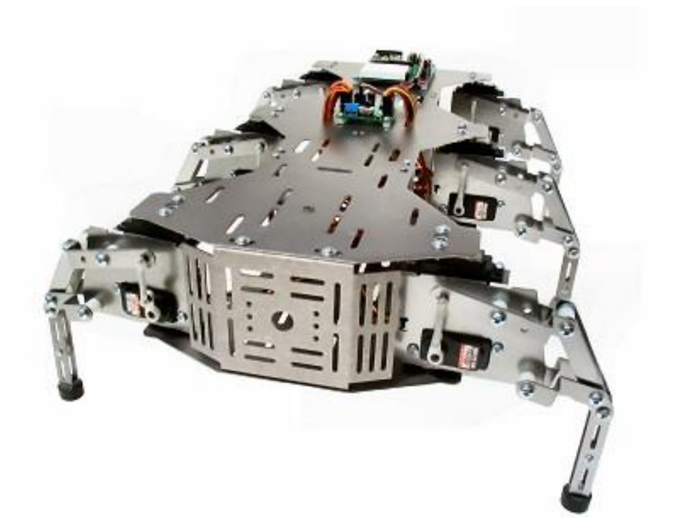


Figure 3 – DEPUSH HexCrawler 1.0

2.3.2 Electrical System

All of the electrical components on the HexCrawler robot are powered using either a 6 volt tethered cable or onboard battery pack, with each of the onboard circuit boards stepping the input voltage down as necessary. P for the Basic Stamp 2 and the Propeller Servo Controller were provided by separate sources due to the fact that while the PSC would consume a very large amount of current (via supply all electrical power), the BS2 would draw very little power. With two separate power sources, the chances of a failure of the main controller input power based upon the possible stall current scenarios of the legs (which would cause voltage to drop, and possible controller reboot) robot were minimized at the expense of greater system complexity.

2.3.3 Software/Embedded Controller

The HexCrawler 1.0 uses the Parallax Board of Education development board with their BASIC Stamp 2 processor. All high-level inputs, outputs, and serial communications were carried out on the BASIC Stamp 2, with the PWM signal generation functionality being relegated to a single 16-channel Propeller Servo Controller daughter board. Communication between the BS2 and the PSC were done via single-wire, RS232 serial communications. Communications with a personal computer were only available via wired RS232 serial communication.

2.4 Technology

2.4.1 Robot/Hexapod Control

2.4.1.1 Microcontrollers

To ascertain which new processor model and development board we would choose for our HexCrawler 2.0 project, we compiled a list of candidates to compare their features, price, and availability in the secondary educational market.

Basic Stamp 2 – Default controller – The BASIC Stamp 2 has a processor speed of 20 Mhz, 32 bytes of RAM, 2KB of Flash ROM, and 18 Input/Output ports.

<u>Advantages</u> – While the raw processor speed is acceptable for most basic tasks, the extremely limited RAM memory of the BS2 prevents us from running code with sufficient variables for inverse kinematics algorithms.

<u>Disadvantages</u> – The BS2 is programming in the BASIC language, which while easy to learn is limited in its real-world scope; nearly all other embedded systems are programmed in the C language.

ARM Cortex-M3 STM32 Family (Alternate Controller 1) – Produced by ST Microelectronics, this processor family is based on the ARM 32-bit Cortex™-M3 CPU. It runs at a processor speed of 72 MHz, has up to 80 I/O ports, either 128Kb or 256 Kb of RAM, eight timers and nine communication interfaces (such as RS232 Serial, Ethernet, SPI and I2C). The processor chips are less than $10 each, and a development board can be purchased $30 and up.

<u>Advantages</u> – Development boards based on the STM32 family are very common, and can be purchased for as cheap as 200 RMB. These boards include features such as JTAG, SD Card readers, TFT Touch Screens, Ethernet, USB and more. The processers are more powerful than the BS2, and are programmed in the C programming language. The STM32 family of processors have better electric power consumption than the BS2, thus making them better suited for mobile robot applications.

<u>Disadvantages</u> – Due to the large number of features, it can be difficult to initialize the various settings and peripherals on the development board unless a software library is utilized.

ATMEL AVR C51 (Alternate Controller 2) – This microcontroller is produced by ATMEL Corporation. It runs at a speed of 33MHz, 256 bytes RAM, has 32 I/O ports and 16 timers. It is programmed in the C programming language. Inexpensive development boards based upon the C51 are common in the educational market.

Advantages – Inexpensive, has a high number of I/O ports, and is commonly available.

Disadvantages – It is much slower and has less memory than the STM32 processor family and only marginally better than the BASIC Stamp 2 in functionality.

2.4.1.2 Technical Support

When selecting an embedded controller for an educational robotics product, the issue of technical support for the chosen processor family is important. When learning through experimental laboratory exercises, there are bound to be conditions, which may necessitate technical support to rectify.

Both the STM32 and AVR C51 are readily and currently available, and thus have the current technical support of their respective manufacturers.

2.4.1.3 Future Expandability

Due to the limited RAM memory available on the AVR C51 processors, a full-featured inverse kinematics algorithm and distributed computing platform may not be feasible. Thus to allow for future growth potential and expandability, the STM32 family is a more desirable embedded controller choice.

Among the many different models in the Cortex-M3/STM32 family, two development board processors were chosen for further development. The first was the STM32F103 and the latter was the STM32F107VC. The prior was in a mostly barebones development board produced by DEPUSH, with a JTAG connector, USB input, RS232 Serial, and two rows of input/output pins as the only features. Because of the lack of additional features, this controller was cheap and was already produced by DEPUSH. To highlight additional features, the Gold Bull (Taurus) development board based on the STM32F107VC was also purchased; this board had the prior features of the STM32F103 plus a 3.2" TFT Touch Screen, Ethernet port, SD Card Slot and more. This board was chosen to display a GUI with diagnostic and feedback information on the touch screen, as well as use the Ethernet port for ease of communication between an embedded controller and nearly any computer.

2.4.2 Robot/Hexapod Locomotion

2.4.2.1 List of Comparable Mechanisms

Due to deficiencies in the mechanical design of the two degree of freedom legs on the current HexCrawler robot in regards to the size of the workspace environment, the addition of a third degree of freedom coupled with a ground-up redesign would optimize the system performance and allow for three-dimensional, straight-line operation (provided it's within the workspace). Before detailed design work began, research into similar walking mechanisms was completed. Two other products offering three DOF legs were found: the RB-Lyn-248 from Lynxmotion Phoenix and the HexCrawler HDATS from DEPUSH.

DEPUSH HexCrawler

HexCrawler	
Company	**DEPUSH**
Material	5052 alloy aluminum, 1/16" thick with clear anodizing
Leg Movement	2 DOF
Dimensions	Body: 49.68 x 40 cm, Height: between 12.3 and 15.2 cm
Cost	$350+

Lynxmotion RB-Lyn-248

RB-Lyn-248	
Company	**Lynxmotion**
Material	5052 alloy aluminum, 1/16" thick with black anodizing
Leg Movement	3 DOF
Dimensions	Body: 19.1 x 14.9 cm, Height: between 5.1 and 13.3 cm, Ground Clearance: up to 8.9 cm
Cost	$261.35+

HexCrawler HDATS

HexCrawler HDATS	
Company	**DEPUSH**
Material	5052 alloy aluminum, 1/16" thick with clear anodizing
Leg Movement	3 DOF
Dimensions	Body: 52 x 49.7 cm, Height: between 12.3 and 15.2 cm, Ground clearance: 12.7 cm
Cost	¥ 10,000 (about $1,500)

As shown above, the RB-Lyn-248 is a very small and compact robot. The HexCrawler and HexCrawler HDATS share many common parts, including the same body. This allows the legs of the two robots to be modular and interchangeable between the two models. However, one trait shared among all three of these robots is that their leg mechanisms do not reach a wide enough workspace volume to make stair climbing a reality. Thus, these three models were used as a reference when designing our new leg mechanism from the ground up.

There were big differences between 2DOF mechanism and 3DOF mechanism from watching video of each robot. 3DOF mechanisms could walk more stable and smoother than 2DOF, and could show more performance in various terrains (pitch, yaw, roll, side to side).

2DOF walking mechanism had no advantages on performance than 3DOF legs. The only advantage of 2DOF walking mechanism was low cost. 2DOF was consisted of only 2 joint which make the leg move vertically and horizontally. As the leg movement was limited, less voltage using was required than 3DOF leg.

Advantage of 3DOF leg was more maneuverable than 2DOF leg. 3DOF was consisted of 3 joint. Including one more joint means it allows variable movement. As one more joint was included, the leg had more flexible and smoother movement. 3DOF allowed vertical and horizontal leg movements and also could move backward and forward. it was possible to be realized some other special movements such as pitch, roll, and yaw by writing the code Therefore the robot could move not only the back and forth but also it could spin and walk side to side like crab.

A Disadvantage of 2DOF walking mechanism was limiting robot movement. This caused the leg contributes to falling sometime. 2DOF mechanism could move only forward and backward and robot movements did not look smooth enough. However, 3DOF needed more voltage to move its legs. 3DOF leg kit cost more than 2DOF, because it needed bigger battery to supply power to move 3DOF leg and needed a better microcontroller, which allowed the mechanism moving in full 6DOF body motion. 2DOF leg also needed only two servos for each leg to operate all possible motion, and 3DOF leg needed three. Therefore, using more servo means it cost more than 2DOF. 3DOF kit also made the mechanism larger. The width of whole body of the robot including leg kit had to be larger than 2DOF.

2.4.2.2 List of Comparable Servos

For the HexCrawler to walk, each joint (degree of freedom) of each of the legs would need to be mechanically powered via a controllable, closed-loop control. There are several means of controlling mechanical leg joints for the HexCrawler project, including servomotors, geared DC motors with position feedback and stepper motors.

Servomotors, commonly referred to as servos, are a small self-contained package involving a DC electric motor, gearbox, position sensor, and feedback adjustment. They are set to a command position via a pulsed-width PWM signal. Servos are common due to their widespread adoption by the hobbyist and educational market, as they are an efficient means of controlling limited movement joints. Servos are available in a variety of limited sizes, torques, and operating voltages. Listed below are the specifications of four specific servos used by DEPUSH for other robots.

Hitech HS-332HD

Hitec HS-322HD	
Description	Three-pole ferrite, nylon bushing
Torque	4.8/6.0v – 42/51 oz-in
Speed	4.8/6.0v – 0.19/0.15 sec
Size	41 x 20 x 37 mm
Cost	$9.99

Hitech HS-485HB

Hitec HS-485HB	
Description	Three-pole ferrite, ball bearing
Torque	4.8/6.0v – 72/89 oz-in
Speed	4.8/6.0v – 0.20/0.17 sec
Size	41 x 20 x 37 mm
Weight	45.08 g
Cost	$17.99

Hitech HS-645MG

Hitec HS-645MG	
Description	Three-pole ferrite, dual ball bearing, metal internal gears
Torque	4.8/6.0v – 107/133 oz-in
Speed	4.8/6.0v – 0.24/0.20 sec
Size	41 x 20 x 37 mm
Weight	55 g
Cost	$39.99

Hitech HS-805BB

Hitec HS-805BB	
Description	Three-pole ferrite, dual ball bearing
Torque	4.8/6.0v – 275/343 oz-in
Speed	4.8/6.0v – 0.19/0.14 sec
Size	66 x 30 x 57.5 mm
Weight	153.09 g
Cost	$39.99

Another alternative option is to design and implement a gearbox for each joint using discrete components and feedback sensors. There are an infinite number of iterations of possible choices for DC motors, gears, bearings and/or bushings, etc. The DC motors can be either pulse-width or variable voltage driven. This allows for a much higher level of customization and specialization, but at the expense of added design, assembly, and debugging time. Designing custom gearbox joints and feedback also necessitates doing a number of feedback calculations on the main robot processor, which can bog down the processor and delay time-critical functions on less powerful embedded controllers.

The last alternative option for joint control of the legs is stepper motors. Stepper motors are commonly found in industrial applications such as CNC machines and industrial robotic arms. Stepper motors work by advancing exactly one fraction of a revolution in a specific direction when one pulse is received. If the actual torque that is seen by the stepper motors *never* exceeds their stall torque, then stepper motors can be used open loop as long as you know the starting position. But since real world conditions may cause one of the stepper motor pulses to be stalled, feedback control is necessary. Using stepper motors on the robot would require purchasing specific stepper motors, gearboxes to increase their useable torque, feedback sensors, and stepper motor drivers.

The most important servo characteristic with respect to the improved HexCrawler leg design was the available torque. While compact size and low cost were desirable factors, the chosen servo would need a minimum torque to control the robot at the specified loads. The two least expensive servos were the HS-322HD and HS-485HB. These servos were small and light, but lacked sufficient torque to control the HexCrawler. Since there was little difference in price between the HS-645MG and MS-805BB servos, additional analysis was necessary. The specifications of the HS-645MG servo were similar to the other inexpensive servos, but had sufficient torque for application in the HexCrawler. While the HS-805BB servo had the highest available torque, its physical dimensions were significantly larger than the other three, standard-size servos. For the final revision of the improved HexCrawler legs, the HS-645MG servos were chosen for all eighteen joints as the ideal compromise between available torque, cost, and physical dimensions.

2.4.2.3 Materials

There are many methods of constructing the structural chassis for prototype mobile robots, including but not limited to sheet metal fabrication, CNC milling and turning, laser-cut plastics and three-dimensional printed plastics.

Sheet Metal Fabrication – Sheet metal designs are characterized by maximizing the physical properties of thin sheets of metal by the addition of flanges and other cold-worked features that strain harden the part. Sheet metal fabrication is widely used in industry due to its fast turnaround and relatively inexpensive manufacturing cost. Sheet metal parts are cut out by one of three methods: turret punch machines, laser cutters, or water jets. Flanges can be bent into parts to add mounting tabs and improve structural properties. Pressed insert nuts can be added to facilitate the assembly process. The 5052-alloy of aluminum is one of the most widely used sheet metal fabrication materials due to its superior combination of cold-working properties, weldability, and material strength over other aluminum alloys.

CNC Machining – Due to the versatility of CNC machines, a wide variety of parts can be fabricated on CNC machines. Features such as through and tapped holes, slots, pockets, and more can be milled into parts. Lathes can turn round parts, such as shafts. Because of the versatility of CNC machines, the cost to machine parts varies depending on part complexity.

Laser Cut Plastics – Many plastics, such as acrylic and ABS, can be used in small laser cutting machines. These machines represent very fast turnaround times for prototyping parts for mobile robots. While the material properties of the chosen plastic may make it unsuitable for some applications, often times there are few problems for small robotics applications.

3D Printing – For very complex parts that would be impossible to fabricate on a CNC milling machine, a 3D Printer can be used to fabricate the part by spraying liquid plastic down one layer at a time until the part is completed. Because it takes time for the part to cure enough for another layer to be sprayed, some parts can take as long as an entire day to fabricate in a 3D Printer. Because of this long time window, 3D printed parts are unsuitable for large-scale production.

2.3.2.4 Kinematics

Inverse Leg Kinematics

For accurate control of a robotic mechanism, closed-loop feedback and inverse kinematics are a necessity. While the closed-loop feedback determines where the robot is relative to where it wants to be and corrects any mistakes, the inverse kinematics are what determine how and to where the robot and its various mechanisms moves.

Inverse kinematics uses the real-world coordinate point of the end effector of a mechanism to back-calculate the necessary angles and/or positions of any variable link in the system to reach the set point, given said point exists within the workspace environment of the manipulator. This allows for the creation of “straight line” motion (via a series of set waypoints along the line), avoiding the common race condition on non-Inverse kinematic mechanism where one link may reach a set end position before the other does, therefore causing an undesirable condition. By setting points, or points along a designated path, the robot is able to move predictably through its environment.

Inverse Body Kinematics

The robot’s whole body is driven by six legs. In one way, the HexCrawler’s gait can be easily designed thanks to the leg IK (Inverse Kinematics) algorithm. On the other way, if the robot is expected to be more vivid, there is another way to apply the IK algorithm. Once after inverse leg kinematics explained, it would be much easier to understand inverse body kinematics.

In body IK, HexCrawler’s six legs would stay on the ground constantly; instead, it’s body’s turn to move. In this way, the robot can show the motion such as rolling, pitching, and swaying. As the dynamics theoretically comes from the 18 servos, the key to body IK is still backwards calculating. The three joints in robot’s one leg are called shoulder, arm, and wrist. When the robot coordinate set up, once the body moves, three shoulders’ positions are determined by the structure dimension. With the leg’s end position known, relationship between servos’ angle and the leg’s status can be expressed by certain equations. Then when a motion is wanted for the robot, shoulders’ position is put into the function with angle output for each servo. In this way, with the ability to obtain various kinds of actions, the HexCrawler would look like a lively dancing creature.

What should be mentioned is that body IK is not a new algorithm. Borrowing the idea from leg IK, the body IK is almost the same as leg IK if you look from wrist to shoulder instead of shoulder to wrist. So there would not be much work besides some transformation in coordinate.

Velocity Kinematics

Besides exact position, velocity is another issue requiring control. But just as the body IK, it's really similar to the original algorithm—leg IK. The principle is still backwards operation. The condition now is legs' or the body's moving speed and the ramp speed of each servo is the result. What needs to be done is differential or vector calculation on the base of position IK.

2.4.3 Distributed Computing and Remote Control

Distributing computing is the means by which sensor data processing and robot control is shared between a less powerful, mobile robot controller and a more powerful, stationary computer. In this arrangement, the less powerful controller typically handles low-level tasks, like getting and setting the necessary inputs and outputs respectively, while the software application on the computer uses its increased processing power to handle computationally intense tasks.

2.4.3.1 List of Communication Technologies

Efficient communication between the robot controller and the computer application is necessary for distributed computing. While wireless operation is desirable for commercial deployment, tethered operation is better suited during the development process. To achieve these goals, there are several common communication protocols available.

Bluetooth

This communication protocol is widely used for low-bandwidth peripherals for computers, cell phones, and other mobile devices. While Bluetooth technology would be unsuitable for real-time video transmission, it can easily handle small file transfers, sensor readings, and robot commands. Communication is carried out between paired devices, thus making Bluetooth a one-to-one networking model. Bluetooth is only available for wireless communication.

TCP/IP Networking (Ethernet, Wifi)

TCP/IP networking via Ethernet, IEEE 802.11 Wireless networks, or CDMA/3G/4G cellular networks is one of the most common communication protocols for computers, mobile devices, servers and workstations. Depending on the networking infrastructure, TCP/IP networks are suitable for either low- or high-bandwidth applications. TCP/IP networks are typically structured in a one-to-many (hub and spoke) network. TCP/IP networking can be either wired (via Ethernet) or wireless (via 802.11 a/b/g/n Wi-Fi or Edge/CDMA/3G/4G cellular networks).

ZigBee

ZigBee is an IEEE 802.15 communication protocol designed for low-power, low-bandwidth devices to be used in a wide variety of applications, such as home- and industrial-automation devices. ZigBee devices can be configured for one-to-one, one-to-many, or many-to-many (mesh networking) communication protocols. ZigBee is only available as a wireless protocol, however if RS232-Zigbee adapters are used wired operation via DB9 serial is possible.

Radio Frequency (RF)

Radio Frequency (RF) communication is typically used in hobby and consumer goods, such as model cars, boats, and airplanes. Radio Frequency communication has no defined standard protocol, with most available products using a proprietary protocol. RF communication for the purposes of controlling a mobile car, boat, or airplane is often one-way, and is regulated to specific frequencies in each frequency. However, these available frequencies often differ between countries.

Infrared (IR)

Infrared (IR) communication is typically used in household appliances for sending commands from the remote control to the base device. Since there are no standard infrared communication protocols, the technology is easily adaptable for most low-bandwidth applications. Infrared communications also suffer from interference, and are only usable in line-of-sight communications.

2.4.3.2 Distributed Computing Hardware

Bluetooth

To enable Bluetooth communication between the mobile robot and the computer application, Bluetooth modems on both ends are necessary. Various breakout boards are available to add Bluetooth functionality to existing robot controllers. On the high-level side, many higher-end computers have Bluetooth built-in. For the computers that lack internal Bluetooth modems, USB Bluetooth dongles are relatively inexpensive.

TCP/IP

Since TCP/IP networking forms the foundation of the Internet and most internal networks, Ethernet and/or 802.11 Wi-Fi functionality is intrinsic to nearly all modern computers, laptops, and mobile devices. On the low-level, some advanced embedded controllers have internal Ethernet capability. From here, a 802.11 Wi-Fi router can be used to achieve wireless operation. For other embedded controllers, Ethernet and 802.11 Wi-Fi breakout boards are available.

ZigBee, RF, and IR

All three of these communication protocols can be added to computers via RS232 serial communications. On the low-end, a RS232 to ZigBee, RF or IR breakout board is necessary. On the high-level side, the same board can be used with a RS232 DB9-USB adapter. Some breakout boards also have internal USB adapters, thus facilitating communication to a computer. However, this precludes use of some devices, such as portable electronics that lack USB ports, from controlling the robot.

2.4.3.3 Distributed Computing Software

Standard drivers for Bluetooth, Ethernet, or 802.11 Wi-Fi would be the only necessary software to communicate via either of these protocols. ZigBee, RF, or IR would all need RS232 serial drivers, along with specific software to initialize, send, and receive data (due to the lack of standard software protocols). These low-level communication drivers would then send standardized data packets to a software application.

Software Application

The software control application that handles the high-level control tasks on the computer can be written in a variety of languages, including Visual Basic, C++, Java, and Objective-C. For versatility, portability, and speed of development, Java was chosen, and a custom application interface would be necessary.

2.5 Current Research

2.5.1 Stair-Climbing Robots

One of the primary project goals was to redesign the HexCrawler walking mechanism, so that it would be capable of climbing stairs. Before this improved design could commence, it was necessary to find the standard stair dimensions from both PR China and the United States (two potential product markets).

	Stair Construction Standard	Thread	Riser	Units
1	International code council	25.4	19.685	cm
2	Chinese standard	28	18	cm
3	U.S standard	25.4	20.32	cm
4	estimated case1	25~27	18	cm
5	estimated case2	26.67	19.685	cm
6	building code(China)	minimum 25	maximum 18	cm
7	building code(U.S)	minimum 27.94	maximum 17.78	cm
8	International building code(IBC)	minimum 27.94	10.16~17.78	cm

As listed above, the single most important dimension was the height of the stair. This is what would determine how much vertical change in elevation the new robot would require between the bottom and top of its travel. In addition, the width of the stair would affect how long the robot body would be. This is to ensure maximum stability when climbing the staircase; a robot two small might find it impossible to climb up, and a robot too large would require more torque, and thus less battery life.

2.5.2 Similar HUST Robotics Projects

To assist the project, similar projects currently or previously underway at the HUST campus involving topics related to robotics were researched. Mao Mingyan, a HUST graduate student, was participating in research in distributing computing between the HexCrawler HDATS and a more powerful personal computer. Her objectives were to:

1. Achieve high-level and low-level communication.

2. Improve the gait movement for the HexCrawler.
3. Combine information from multiple sensors to be used for the HexCrawler HDATS to avoid obstacles.

Mao's plans were to:

1. Initialize serial communications between PC and the AVR C51 processor to control servos
2. Allow for adjustable walking gaits
3. Create a modular robot sensor debugging system
4. Install sensors
5. Test HexCrawler HDATS walking gait when suspended in the air
6. Test and debug HexCrawler HDATS walking gait code when on ground
7. Enhance the robot.

Hexapod chassis

DEPUSH provided Mao with a HexCrawler HDATS – a version of the HexCrawler with 3DOF legs.

Obstacle Avoidance

One of the main tasks was to install some sensors to add the function of obstacle avoidance.

Sensor sub-module was debugged to ensure the robot can use all the information from surroundings.

Distributed Computing

The upper and lower computing communication was achieved. The upper level was PC, and the lower level was C51 Board. Serial communication was sent to PC by using the advanced C51 Board.

Progress

The movement of obstacle avoidance can be accomplished when the legs were not touching the ground.

Current challenges

The measuring range of the QTI sensor provided by DEPUSH was narrower than expected, so the sensor might return wrong signal sometimes.

Chapter 3: Project Objectives

Based on the meeting held with DEPUSH managers and engineers, the following list of objectives was compiled to measure the success of the HexCrawler improvement project. Each point is comprised of a general statement to provide direction and a quantitative statement to assess improvement over the original HexCrawler.

1. HexCrawler 2.0 should have improved mobility over the original HexCrawler. The robot's body should have a full 6 Degrees of Freedom (DOF) while keeping all of its feet in a fixed position. More specifically, the body should be able to translate side to side, forward and backward, and up and down, as well as change pitch, roll and yaw, or any combination of the previously listed movements.

2. The redesigned HexCrawler should have increased stability and smoothness while the gait is in motion. Explicitly, the maximum level of jerk, or fourth derivative of position with respect to time, of HexCrawler 2.0 should be less than that of the original HexCrawler.

3. The new HexCrawler should have terrain adaption capabilities. This will allow it to traverse irregular and uneven ground without causing abrupt disturbances to the orientation of the robot's chassis or its gait.

4. HexCrawler 2.0 should have the capability to traverse up and down standard staircases. This is an extreme case of the terrain-handling feature listed above. In this instance, the robot should adjust the pitch of its body to match that of the stairs.

5. Even with the added features and capabilities, HexCrawler 2.0 should not be excessively expensive to produce. DEPUSH sells the original HexCrawler for ¥10,000; their cost is approximately ¥7,000. HexCrawler 2.0 should cost no more than ¥7,000 to produce.

Chapter 4: Methodology

4.1 Benchmark Testing

Keeping in mind that the HexCrawler 2.0 project was an effort to make improvements to the original HexCrawler design, it was imperative that baseline parameters were established so that any enhancement in the new design could be quantified. The team measured basic dimensions of the original HexCrawler's chassis and legs. Velocity and payload capacity were also measured. The fourbar linkage leg-mechanism was simulated using CAD software so that its kinematics could be analyzed. The benchmark testing also helped the team to identify the weaknesses of the original HexCrawler so that the project's objectives could be more clearly defined.

As an additional level of analysis, the original HexCrawler was modeled in its entirety in SolidWorks. With this powerful tool, weaknesses in the geometry of the original design could be quickly identified and analyzed. The entire CAD model was also exported to ANSYS so that finite element analysis could be conducted on its structure.

4.2 Designing HexCrawler 2.0

4.2.1 Mechanical Design

Following the results of the benchmark tests conducted on HexCrawler 1.0, it was determined that the poor geometry and limited range of motion of the leg linkage were the primary reasons that stability and smoothness of the robot's gait were less than ideal. As a result, redesigning the HexCrawler's legs became a priority of the mechanical team. After several iterations, an optimized leg geometry was determined that could offer the mobility that a six DOF body would require as well as the increased range of motion that the terrain adaption objectives dictated.

The new three DOF leg offered a superior workspace as compared to the original HexCrawler's two DOF leg, but was also far more rigid due to improved sheet metal construction. In order to power the new legs, servos with improved torque output were chosen. This helped to increase leg speed and payload, but also made it possible to position the foot very precisely.

In order to accommodate the added degree of freedom on each leg, the robot's chassis had to be redesigned as well. Using the same tabbed-sheet metal construction methods that were employed on the legs, the body was made to be more rigid than the one on the original HexCrawler. The team made ease of manufacturing and assembly a priority in order to reduce cost and make the robot accessible to a wider customer base.

4.2.2 Control System Design

DEPUSH managers' requests to make the improved HexCrawler more mobile and able to adapt to different terrains necessitated that a more power control system be implemented. Company managers, Zhang Huiping and Zhou Jie, recommended that the team develop the control system around an ARM-based processor because it was more accessible in China than the Basic Stamp 2 controller on the original HexCrawler. The ARM processor they recommended was orders of magnitude more powerful than the BS2 processor. It was capable of performing all of the necessary calculations required for advanced kinematics algorithms and had enough resource overhead to implement new features in the future.

The control team decided that an inverse-kinematics engine would be required for HexCrawler 2.0 in order to implement six DOF body control and terrain adaption. This code was based on open-source kinematics code written by Jeroen Janssen for a similar hexapod design. A modular software package was designed so that new functionality, such as sensors, could be added easily in the future. As an added feature, a high-level, computer-based, Java control interface was designed so that users could easily interact with the robot via an Ethernet connection.

4.2 Testing

Due to time restrictions on the project, the team was not able to fully test the HexCrawler 2.0 design. Only seven weeks were allotted for the design, construction, and testing of both the mechanical and control systems of the improved robot.

The team was able to confirm that the dimensions of the body of the new robot were almost identical to that of the original HexCrawler. The low-level control system implemented allowed users to control each of the six degrees of freedom simultaneously. Given more time, the team could have quantifiably substantiated what were obvious improvements over the old design.

4.3 Cost

The designers on the team kept cost-effectiveness in mind when creating HexCrawler 2.0's chassis. An effort was made to minimize the number of discrete parts needed to complete the design. In addition, finite element analysis was used when iterating the model to ensure that the final product was not over-designed. The team made sure to keep a detailed list of all parts and components procured for final assembly of HexCrawler 2.0 so that an accurate Bill of Materials could be delivered to DEPUSH.

Chapter 5: Results

5.1 Introduction

Our methods and tasks for the HexCrawler improvement project were motivated by DEPUSH's wish to create a versatile hexapod robot with ground adaption capability and a wireless, high-level control system. The results section is organized according to our methodology and research of hexapods and related technologies throughout the project. After arriving at HUST, the team met with DEPUSH management to ascertain concrete specifics on the educational robotics platform to be designed during the two-month project. For this undertaking, the robotic system required a complete redesign from the ground up; this is a large undertaking for any group during such a limited time span. The group split into two teams: one to concentrate on designing a robot chassis capable of meeting and exceeding DEPUSH's expectations, the other to implement a control system capable of managing the new mechanism.

First, the team performed a mechanical analysis of the existing HexCrawler robot, making sure to note which aspects of the design were the limiting factors in terms of the system's overall capabilities. After determining the leg mechanism was a primary detractor, the team set out to design three degree of freedom legs that could ultimately give the robot's body a full six degrees of freedom.

After finalizing the mechanical design, the control team implemented a complex inverse kinematics engine on an ARM-based microcontroller. Once this new "low-level" control system was functional, a "high-level" Java interface was designed and prototype. Based on the results of our work, we formed our conclusions and recommendations for future work.

5.1 Mechanical System

5.1.1 Modeling and Benchmarking HexCrawler 1.0

The first task for the mechanical team was to model and analyze the design of the old HexCrawler. The results can be seen in Figure 4. Using this model as a reference, the team was able to determine that the fourbar leg mechanism limited vertical foot travel to 2cm. Although the mechanical advantage provided by the crank-rocker configuration on leg reduced the torque requirements for the servo, it also severely limits the mobility of the body.

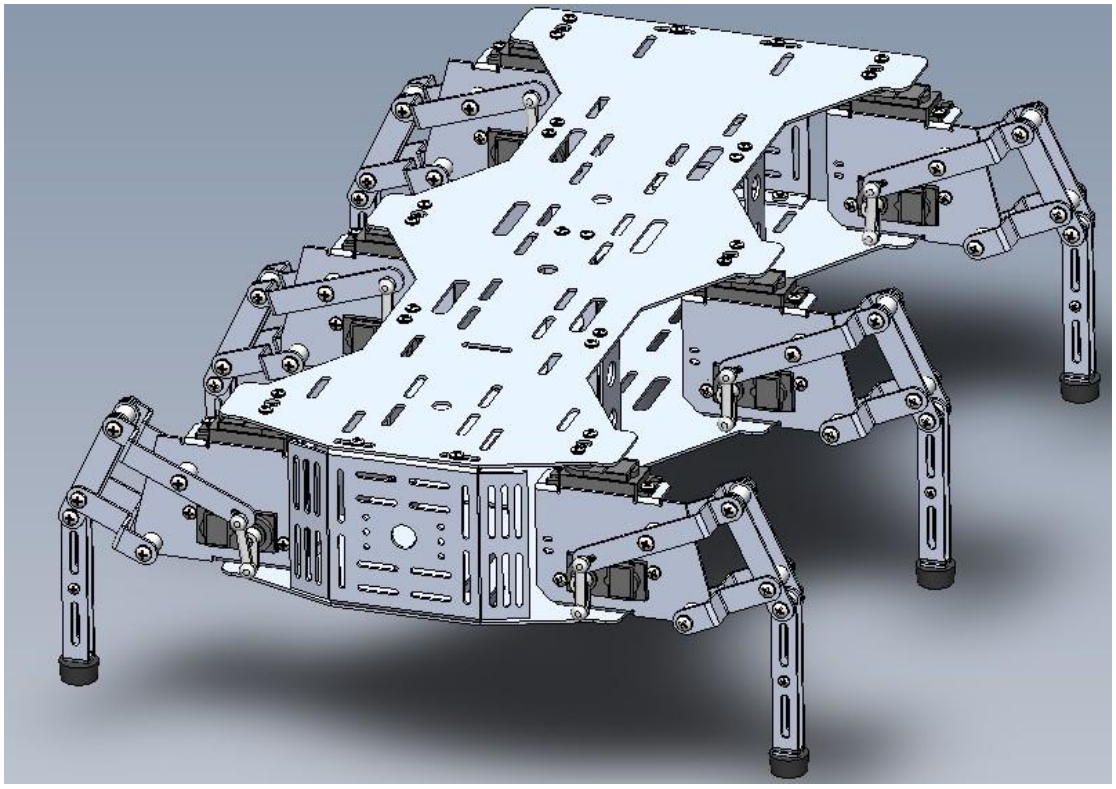


Figure 4 – SolidWorks model of HexCrawler 1.0

The only full degree of freedom occurs when the body rolls from side to side. Rotations about either of the remaining two axes, or translations about any of the three primary axes require two or more of the feet to slip. A sliding joint such as this is classified as having only a half-degree of freedom. Therefore, it was concluded that HexCrawler 1.0 had only 3.5 degrees of freedom (five half degrees of freedom plus the one whole degree of freedom.)

After the CAD modeling was completed, individual parts and the entire robot assembly could be exported for finite element analysis in ANSYS. Using ANSYS, stress analysis was conducted on individual components and assemblies to determine the weaknesses in the original design. It was determined that the many of the linkages within the leg were susceptible to torsion due to the high number of cantilever loads in the mechanism (see Figure 5.) In order to improve upon the design, the cantilever loads were removed by designing brackets that surrounded the leg servos and distributed the load on both sides.

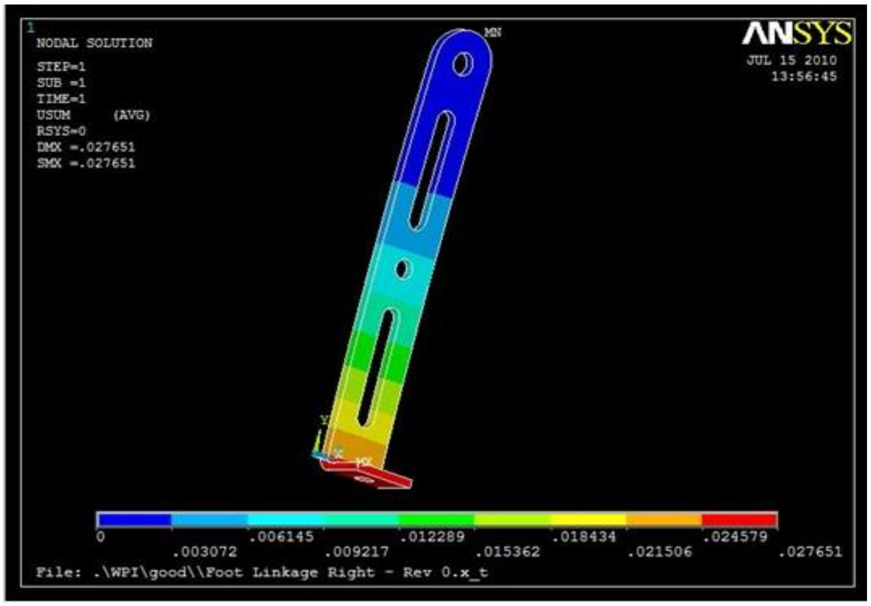


Figure 5 – Stress analysis of a leg linkage using finite element analysis in ANSYS

5.1.2 Benchmarking the Existing Robot

The next step in the analysis process was to perform benchmark tests on the existing HexCrawler robot. From these tests, a set of criteria could be created with which to judge the redesigned HexCrawler's performance. Simple yet effective tests were designed to measure the robot's velocity and stability while the gait was in motion. The velocity was measured by letting the robot walk a known distance and recording the time elapsed. Velocity is simply the distance divided by the time as shown in the following equation.

$$velocity = \frac{distance}{time} = \frac{120\ cm}{23\ s} = 5.2\frac{cm}{s}$$

To determine the stability of the original HexCrawler while in motion, a 3-axis accelerometer was connected to an STM32 microcontroller to take continuous measurements and send the raw data to a PC for later analysis. In this test, x-axis represents forward and reverse accelerations, the y-axis represents lateral accelerations, and the z-axis represents vertical accelerations. Acceleration data on its own cannot directly reveal the stability of the robot. For example, an object can have a high, constant level of acceleration and be perfectly smooth. Rather it is the change in acceleration over time, or "jerk," that should be calculated in order to determine how smooth something is. In order to measure the jerk during the robot's gait, acceleration data was recorded at 10ms intervals and the derivative was computed at those discrete intervals in Matlab. Statistical analysis was then conducted on the jerk data to make it easier to compare the results to the data from the new design.

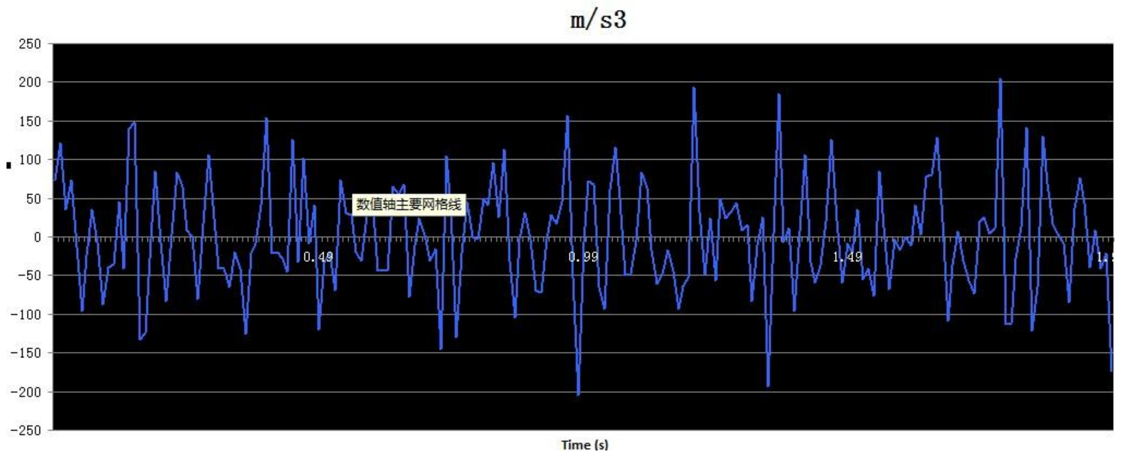


Figure 6 – Jerk (m/s^3) measurements along the x-axis with respect to time (seconds)

Table 1 – x-axis jerk statistical data

Abs Average	52.47797853
Average	-0.11872846
Max	243.618597
Variance	4586.99002
Standard Deviation	67.72732107

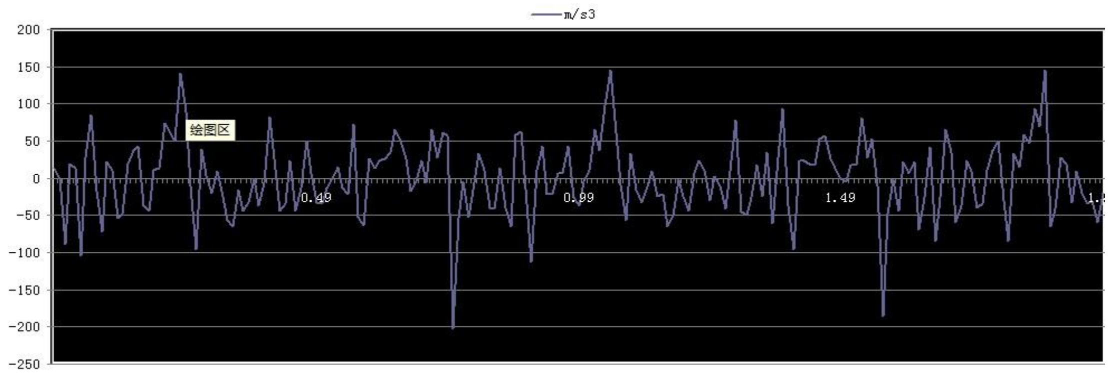


Figure 7 – Jerk (m/s^3) measurements along the y-axis with respect to time (seconds)

Table 2 – y-axis jerk statistical data

Abs Average	38.97626889
Average	-0.077670393
Max	247.900662
Variance	2632.089819
Standard Deviation	51.30389672

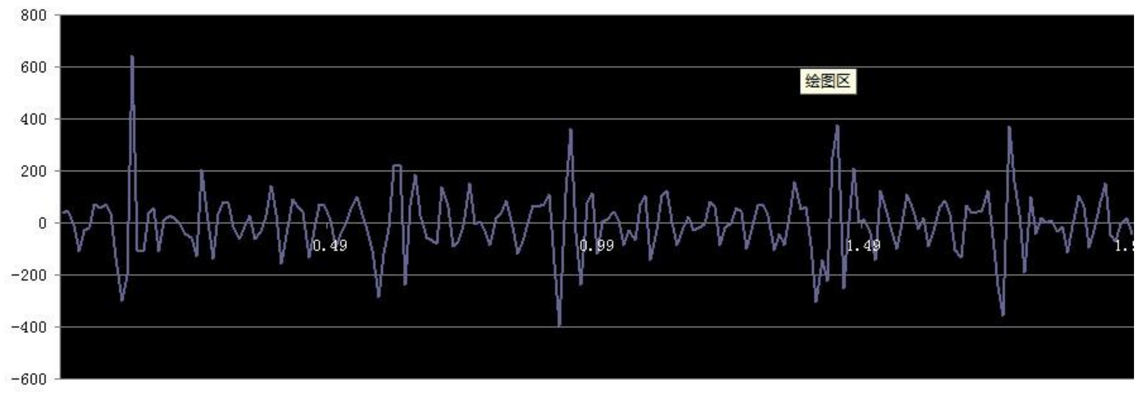


Figure 8 – Jerk (m/s^3) measurements along the z-axis with respect to time (seconds)

Table 3 – z-axis jerk statistical data

Absolute Average	84.93371891
Average	0.021115646
Max	644.073588
Variance	16187.20113
Standard Deviation	127.228932

5.1.3 Defects in the Original Design

The original HexCrawler was designed approximately ten years ago as a low-cost educational hexapod. In order to meet the requirement that the robot should be able to support its body weight and extra payloads while using less-powerful, HS-322HB servos, torque output is multiplied by taking advantage of the inherent mechanical advantage created by the fourbar linkage in each leg. Although this provides a cost savings, the downside to the mechanical advantage is that range of motion is significantly limited. Today, servos are more readily available, and although they are still one on the more costly components in a robotic system, they are far less expensive than they were ten years ago. Now the trend is towards capabilities, not cost-savings. Now, with more powerful servos such as the HS-645MG available to roboticists and hobbyists, it is possible to increase the range of motion by removing the fourbar linkage. The torque output of the HS-645MG is sufficient such that it can drive the leg joints of the new HexCrawler directly while still supporting an ample payload.

The limited range of leg motion of the original HexCrawler is because the foot segment is an extension of the coupler of the fourbar mechanism. The foot must follow a very restricted curvilinear path known as the coupler curve. In addition, because the relationship between the servo angle and the output angle of the mechanism is non-linear, foot position can only be approximated.

With a three degree of freedom leg, the foot can be positioned precisely at an infinite number of points in a three-dimensional workspace around the shoulder. The size of the workspace is only limited by the length of the leg segments and the mechanical limits of the servo movement. Therefore, it is necessary to utilize a three DOF leg to obtain six DOF body movements and a more precise foot positioning. The only downside to the use of three DOF legs is that it requires six extra servos than the older two DOF design. In order to control all 18 servos, a more complicated control system, as well as a larger power source is needed.

The team has researched several three and four DOF leg designs to use as inspiration for the HexCrawler 2.0. The figures below illustrate a few examples of other multi degree of freedom leg designs.

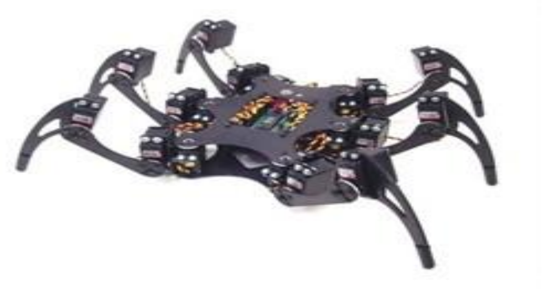


Figure 9 – Phoenix - three degree of freedom legs

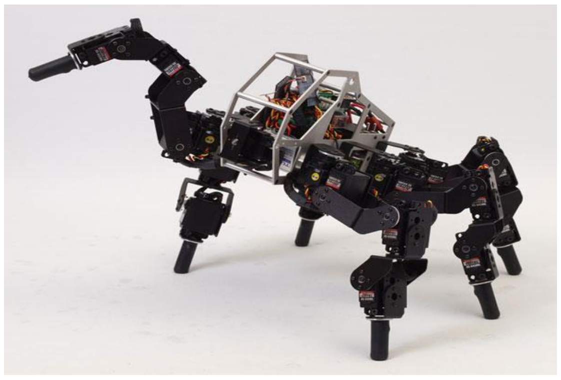


Figure 10 – T-Hex, four degree of freedom legs

Both hexapods pictured above are current designs that use at least three servos per leg. Also, notice that neither of the robots above make use of fourbar mechanism. This greatly simplifies the overall structure of the hexapod.

5.1.4 Designing 3 DOF Legs

One of the first things to consider when designing the new legs was to choose appropriate servos. The original HexCrawler robot used HS-322 and HS-475 servos. However, the robot was designed ten years ago. The new robot should utilize servos with higher peak torque, especially if the ability to climb stairs is desired. After researching Hitec's current line of servos, the team considered four servos: the HS-322HD, HS-485HB, HS-645MG and the HS-805BB. The operational specifications for these are provided in Section 2.4.2.2 above.

Our team decided to use the HS-645MG servos for two reasons: One is that they output the maximum amount of torque for the given size packaging, the second is that DEPUSH has that particular model readily available to them.

After comparing the new three DOF legs with the two DOF legs on the existing HexCrawler, the difference in capabilities is obvious and impressive. The mechanism allows for very precise foot positioning which makes the robot's gait noticeably smoother and more stable.

Considering the possible applications of these kinds of education robots in the future, ease of assembly and manufacturability was a priority when designing the new legs. The entire robot was designed so that it used the fewest number of discrete parts possible to reduce manufacturing costs and the chance of mistakes during assembly. In addition, each leg was designed to be interchangeable to reduce further confusion during construction.

Another important feature of the redesigned robot legs is that they do not make use of any fourbar linkages. Each HS 645MG servo provides enough torque to drive a joint directly. This creates a relationship, in polar coordinates, between the center of the servo horn, and the tip of the leg segment. This is the driving idea behind kinematics calculations.

The mechanical design team made sure to add more mounting slots and holes than the previous HexCrawler had. Although these features increase the complexity and cost of manufacturing, the overall benefits outweigh their costs. The increased number of mounting slots not only adds mounting options for new control boards and sensors, but they reduce the weight of the robot without compromising its structural integrity.

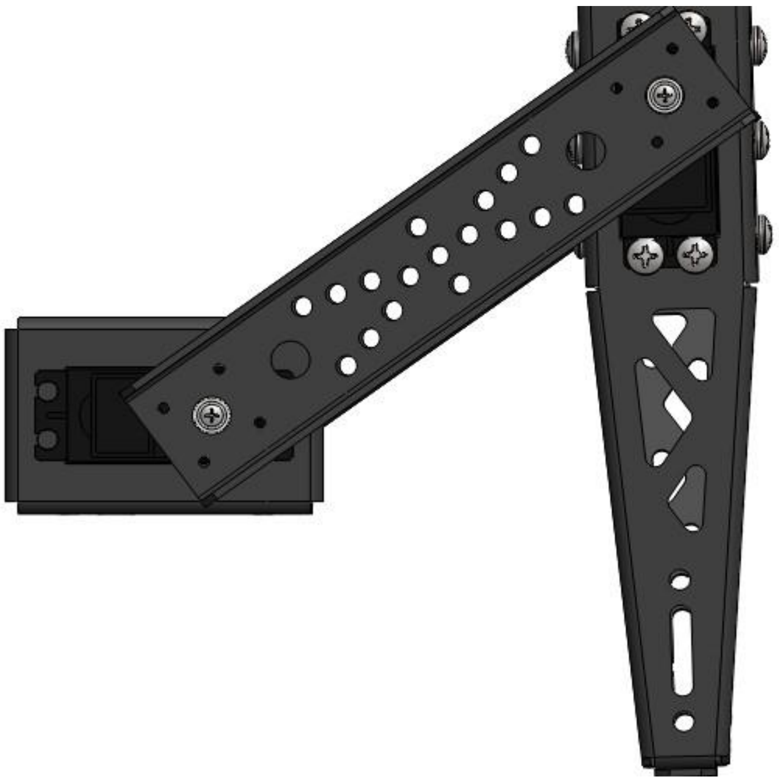


Figure 11 – HexCrawler 2.0 Leg Assembly

5.1.5 Designing the Chassis

In order to accommodate the new three DOF legs, the chassis had to be redesigned to incorporate the extra servos. The chassis of the HexCrawler had two immediate problems: it was not rigid enough, and it lacked proper access for components.

Another important feature of the redesigned HexCrawler is that although the body of robot consists of two separated plates, the upper plate of these two has two symmetry holes of which the area is bigger than its metal area. The significantly improved redesign of the HexCrawler chassis is intended for two purposes. One is making it more convenient for users to place PSC board, STM32 board, and current circuit board into the robot's body easier without having negative effect on the appearance of the HexCrawler.

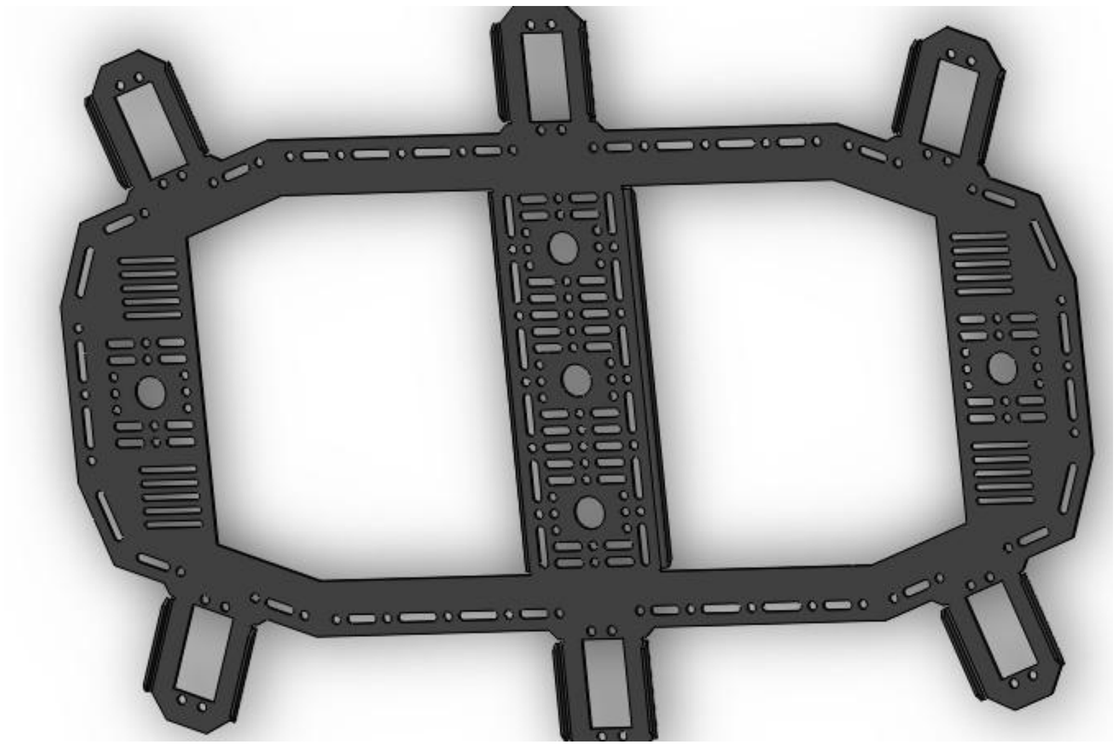


Figure 12 – HexCrawler 2.0 Top Plate

The other plate of those two plates also has impressed appearance for possessing a myriad of holes and slots on it. Considering future extended function such as sensing the distance from the robot to the obstacles, avoiding obstacles by adapting robot intelligence visual technology with camera and remote control with Bluetooth, designers and managers of DEPUSH Company maintained that it is indispensable for the redesigned HexCrawler to have enough amounts of slots and holes as well.

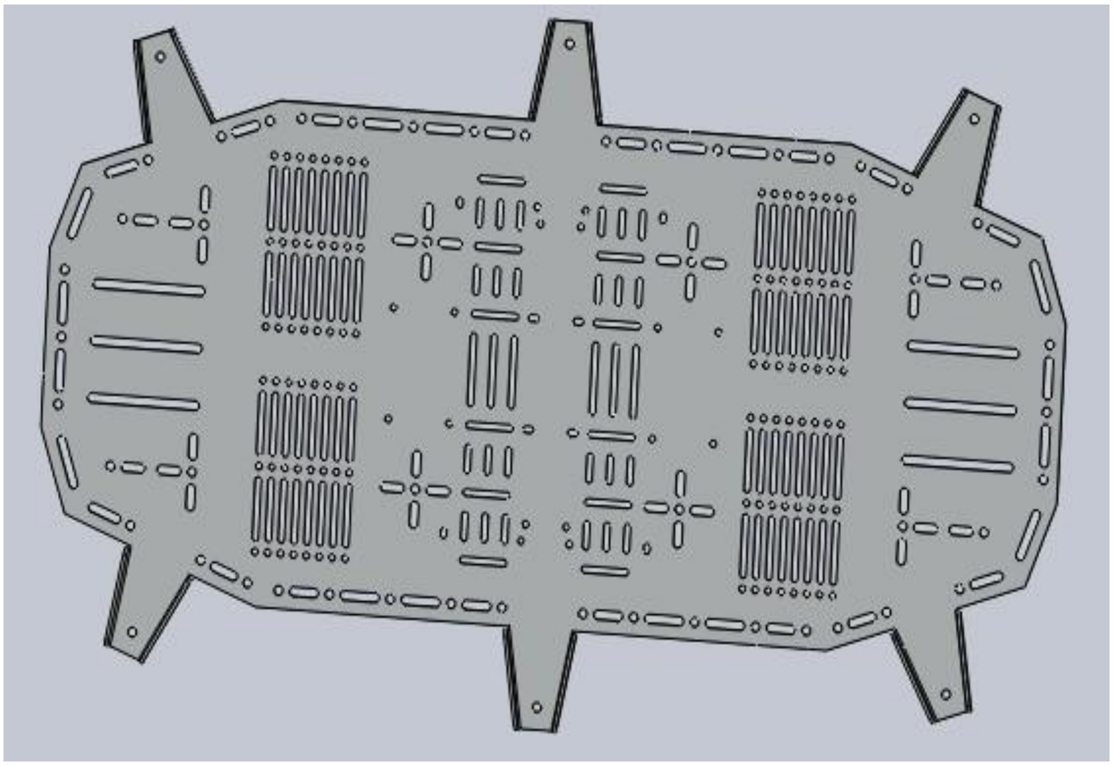


Figure 13 – HexCrawler 2.0 Bottom Plate

5.1.6 Manufacturing and Assembly

Like the original HexCrawler robot design, the new one uses 5052-allow aluminum sheet metal for all structural components. However, to adapt to metric standards, all components were redesigned to integer millimeter sizes (wherever possible), with the material thickness of all the new components being set to 1.6mm gauge. The surface of all sheet metal components was covered in a protective, black powder-coated finish. Besides protecting the parts from corrosion in extremely humid weather, the black powder-coating finish helps the product look more refined and professional, both positive traits that consumers look for in products.

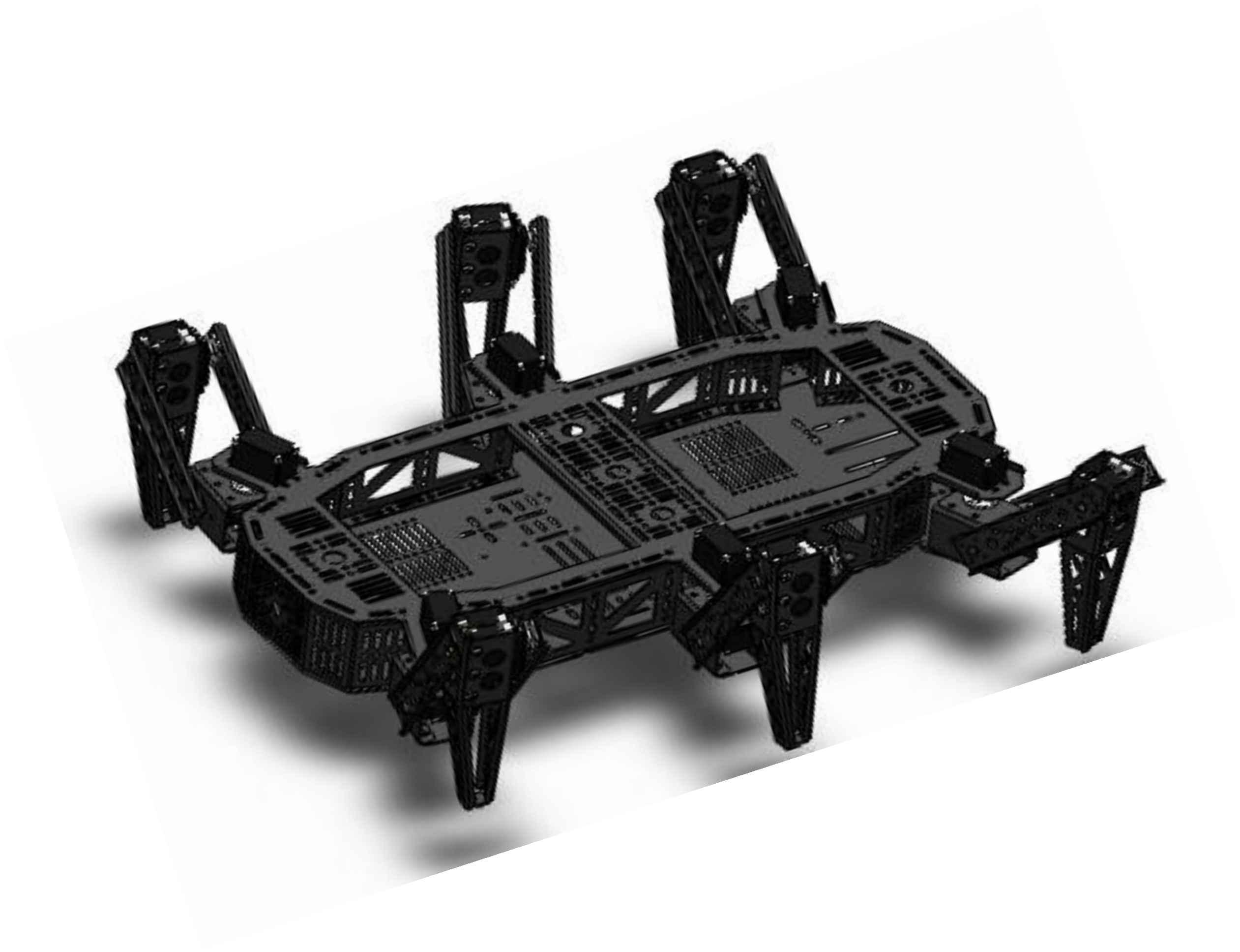


Figure 14 - CAD Rendering of HexCrawler 2.0

5.2 Control System

On the support of the robot's body structure, control system could be the heart of the robot and enable it to have strong ability not only in movement but also in other various functionalities. In this part, main work was the transformation from original controller to a new one. Besides, coding on new controller and testing sensors was also parallel work.

5.2.1 Analysis of BASIC Stamp2 and PSC

The original HexCrawler beard the board (showed in figure xx) having a microcontroller on it called Basic Stamp 2 (BS2), where was the beginning of research in control system. The board was designed and made by Parallax Inc., which integrated some interfaces but was not enough or suitable for the function extension expected in this project.

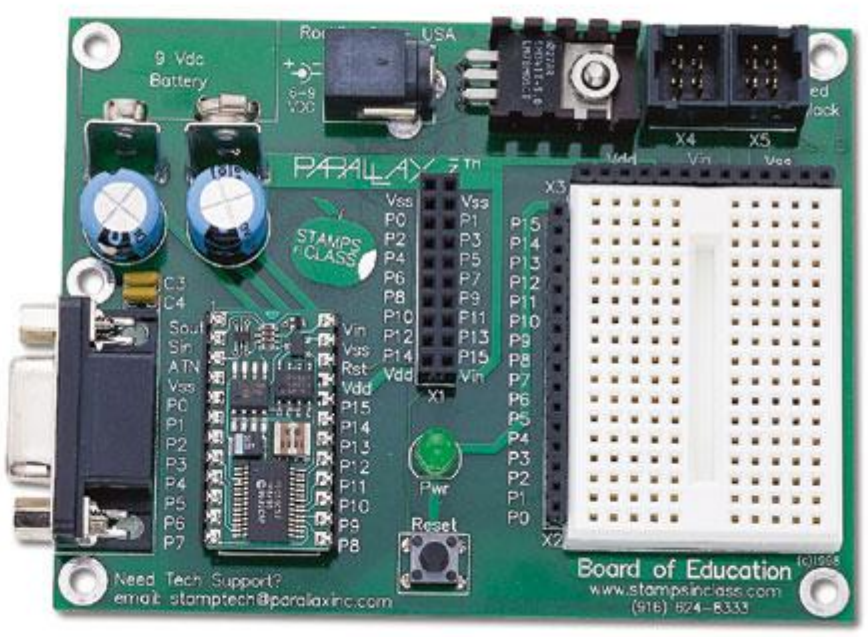

9 Vdc Battery
PARALLAX
STAMPS in CLASS
Vss P0 P2 P4 P6 P8 P10 P12 P14 Vdd
Vss P1 P3 P5 P7 P9 P11 P13 P15 Vin
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
Pwr
Reset
Board of Education
www.stampsinclass.com
(916) 624-8333
Need Tech Support?
email: stamptech@parallaxinc.com

Figure 15 - Basic Stamp 2, Board of Education

5.2.2 Control Board

The original board was not complex, with simple programming language structure. Even though it would be convenient to use this board since many sensors in hand were also designed and manufactured by Parallax Inc. On the other hand, the simplicity made it too integrated, which mean it would be hard to extend its ability. Because of the inverse kinematic contained in the code, its poor calculation capacity was another reason it was abandoned.

5.2.3 Propeller Servo Controller

Before the new controller was discussed, an important component should be mentioned. Besides the heart, some “nerves” was also required if the robot wanted to move. So the original robot had Propeller Servo Controllers (PSC) to directly drive servos. The PSC acted as a sub-commander and amplifier facing servos in front of the BS2.

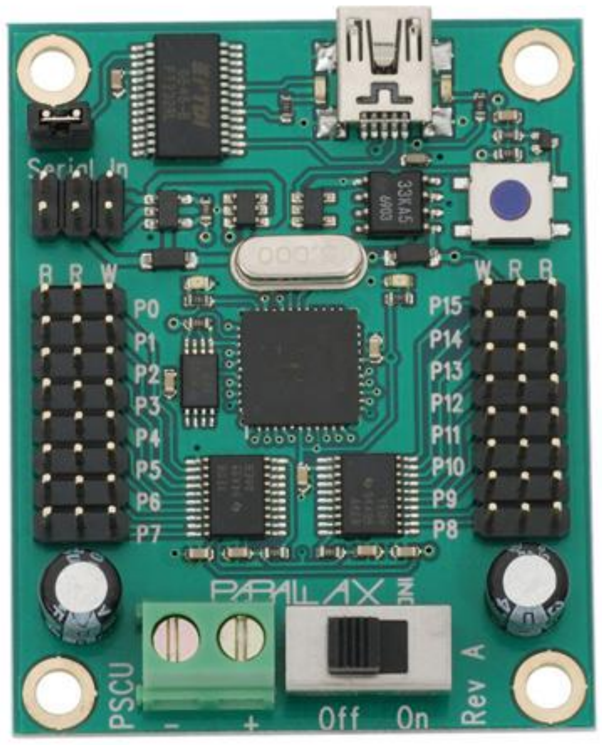


Figure 16 - Propeller Servo Controller

The PSC supports several commands that are sent to it via serial protocol. These commands can come from the TTL serial interface or the USB port. Both serial inputs run at 2400 bps by default at startup and can be switched to 38.4 kbps by sending a command to the PSC. The PSC does not support auto baud with the default firmware installed. The data must be sent non-inverted (true) using 8 data bits, no parity and 1 or 2 stop bits.

The syntax of command sent to the PSC is the same. No matter which controller is used, signal has to be sent through serial line in certain format. Each command is preceded with an exclamation point (!) and the letters, "SC", so every command will appear as "! SCxxxx" $0D, where "! SC" is the preamble, "xxxx" are the command/parameter bytes and $0D is a trailing carriage return. Every command message contains exactly eight (8) bytes including the carriage return. Taking position command as an example, the syntax is as followed:

"! SC" <channel> <ramp speed> <low byte> <high byte> <CR>

Reply: None

To move a servo to a location you must write a position command to the PSC. Each position command is comprised of the preamble, the selected channel of servos, the ramp speed (moving velocity), low byte/high byte of the pulse width (position degree) in 2 μs increments and a carriage return ($0D). The preamble is "! SC". The channel is a byte value from 0 – 15 (16 – 31 if this is the second unit in a network). The ramp speed is a byte value from 0 – 63 that controls the speed the servo moves to its new position.

In this project, 18 servos had to be controlled, but one PSC chip could only do 16. In this case, two PSC chips were connected.

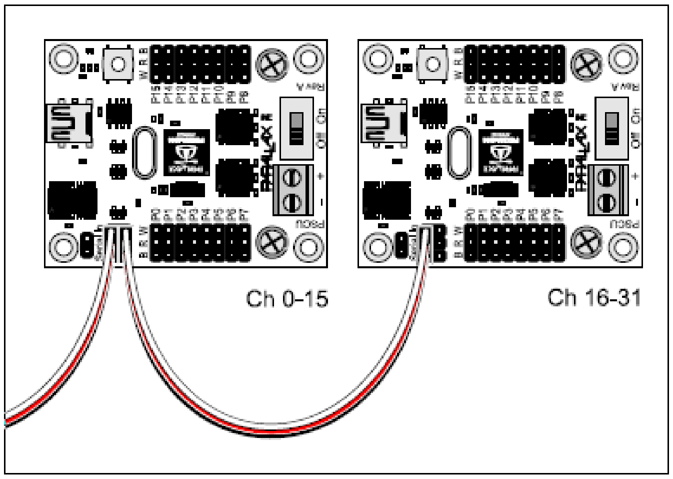


Figure 17 - Two PSCs Connected to Control 18 Servos

The former board had walking forward code readily written. The gait it applied is called triangle gait. The six legs were separated into two groups forming two triangles. In the figure below, leg 1, 3, 5 are in one group and others in the other group. When the robot wanted to move, its two groups of legs would move one by one, but synchronized in one group. This gait was an outcome of bionics which resembled some insect's moving gait. Although there were other gaits such as ripple gait or wave gait, this was the mostly used one by natural creatures since it was one of the most stable one even when moving fast.

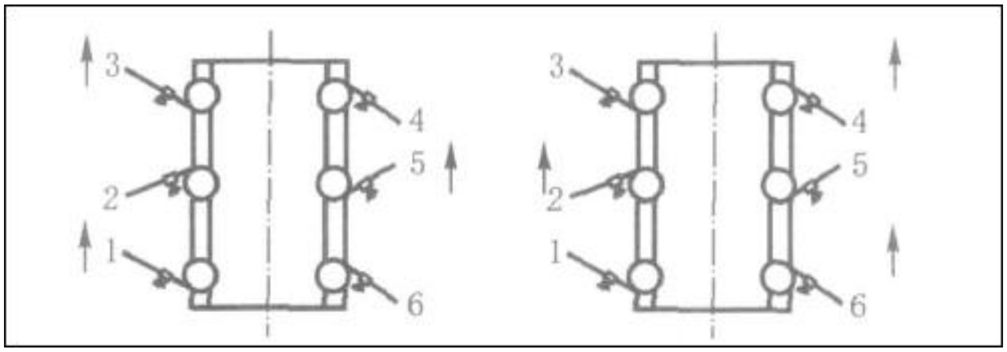


Figure 18 - Triangle Gait

When the ramp speeds of these two groups were set to the same value, the hexapod could move straight forward (not considering mechanical factors leading inaccuracy). Then the robot could easily obtain the desire to turn left or right just by setting right triangle faster than the left or on the opposite.

Visually could be seen is that the former hexapod moved jerkily. Actually in order to test if the triangle gait worked well in the former robot, accelerators were used to measure acceleration.

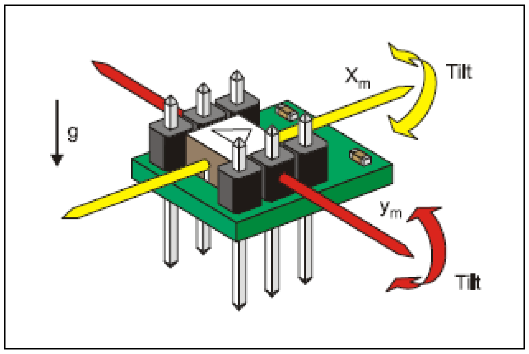


Figure 19 - Accelerometer Orientation

Since the forward-backward direction was the least sensitive direction of shaking, the other two directions were chosen being tested. In the following figures, it was clear that the accelerations in these two directions were changing severely, which corresponded to the visual phenomena.

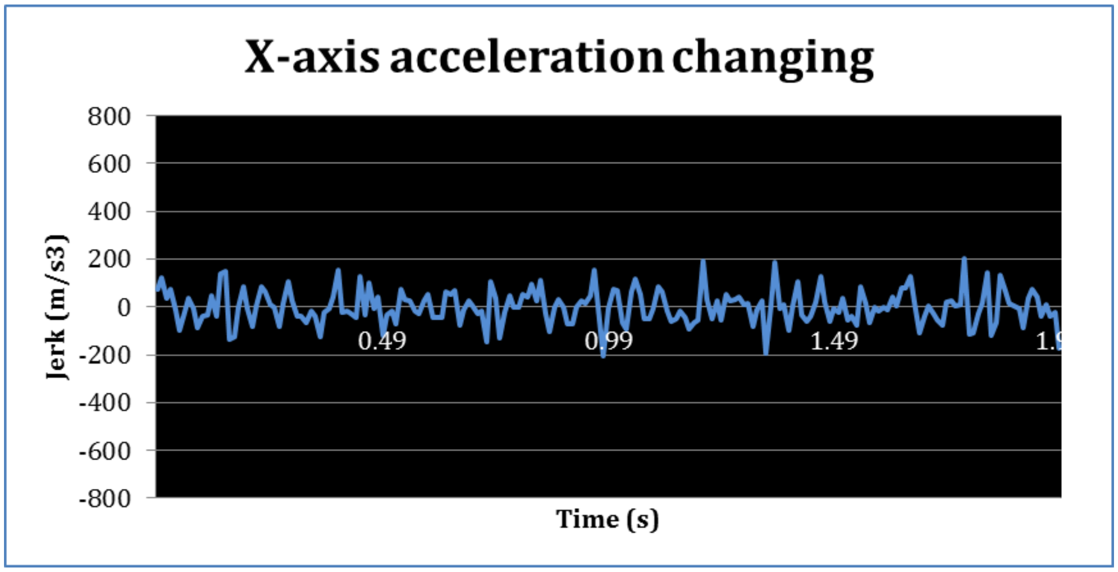


Figure 20 - X-Axis Acceleration

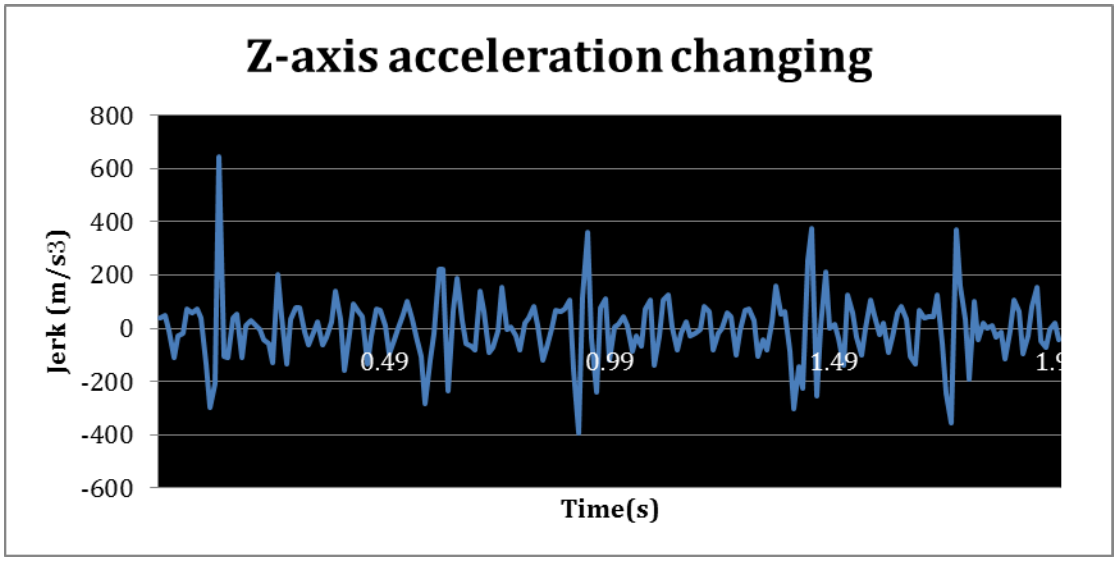


Figure 21 - Z-Axis Acceleration

Through analysis, it was estimated that there were several reasons for the bad performance. The first one was the bad design of legs. And since it had been nearly ten years after it became a product. The bad performance was a poor foundation for further application and development. For example, when it was required to sense object or form image through camera while it moved, the jerky movement would lead the image shaking and make it difficult for image treatment such as sensing obstacle and avoiding.

To sum up, there was no reason for continuing exploit on this base. That's the reason for this project to choose a new powerful controller, design or purchase a new suitable board, redesign a smooth gait for new mechanical structure (basically triangle gait).

5.2.4 Choosing an Alternate Controller

According to different chips' performance, costs, team's academic background, ARM processor was chosen (the reason has been mentioned before in the technology part of this paper).

Firstly, DEPUSH provided an educational board with an ARM processor—STM32 103RB on it. Since it had few interfaces expected to support various application, it was decided to purchase a new board not only because it had a better controller STM32 107VC processor, but also for the various interfaces it provided: touch screen, SD card slot and so on. Before the new board arrived, the work mainly focused on the research of the STM32 103RB board.

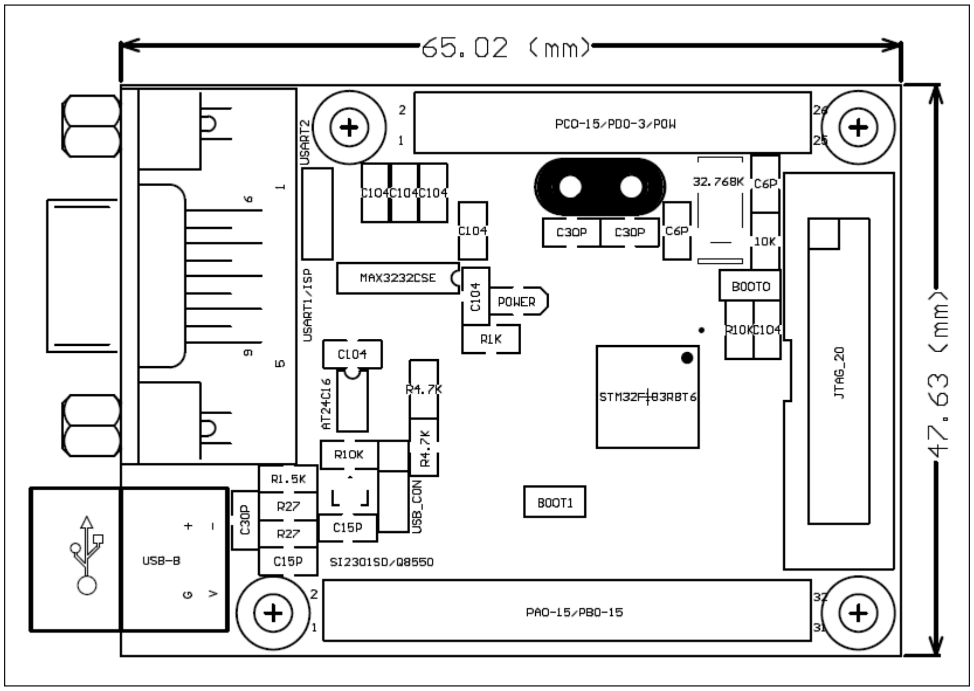


Figure 22 - DEPUSH STM32 Development Board

The ARM was a 32-bit reduced instruction set computer (RISC) instruction set architecture (ISA) developed by ARM Holdings. It was known as the Advanced RISC Machine. The ARM architecture was the most widely used 32-bit ISA in terms of numbers produced. The relative simplicity (compared to advanced processors) of ARM processors made them suitable for low power applications. This had made them dominant in the mobile and embedded electronics market as relatively low cost and small microprocessors and microcontrollers.

The ARM was powerful enough for application in such level as the robot in this project. But it was hard for team members in the project to start off since the way ARM processors work was hard to understand to people lack relative experience.

ARM had 31 general-purpose 32-bit registers. At any one time, 16 of these registers were visible. Users could change the value in these registers to control certain function, read working status and communicate data with other devices. The other registers are used to speed up exception processing. All the register specifiers in ARM instructions can address any of the 16 visible registers.

The main bank of 16 registers is used by all unprivileged code. These are the User mode registers. User mode is different from all other modes as it is unprivileged.

5.2.5 GPIO

General Purpose Input & Output (GPIO) was a fundamental function STM32 processor provided. It was one of the basic ways for serial communication.

Each of the general-purpose I/O ports has two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH), two 32-bit data registers (GPIOx_IDR, GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 16-bit reset register (GPIOx_BRR) and a 32-bit locking register (GPIOx_LCKR), in which x represents the number of GPIO.

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words (half-word or byte accesses are not allowed). The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIO registers. This way, there is no risk that an IRQ occurs between the read and the modify access.

Taking the GPIO acting as an input as an example, the configuration order was as followed:

- The Output Buffer is disabled
- The Schmitt Trigger Input is activated
- The weak pull-up and pull-down resistors are activated or not depending on input configuration (pull-up, pull-down or floating)
- The data present on the I/O pin is sampled into the Input Data Register every APB2 clock cycle
- A read access to the Input Data Register obtains the I/O State.

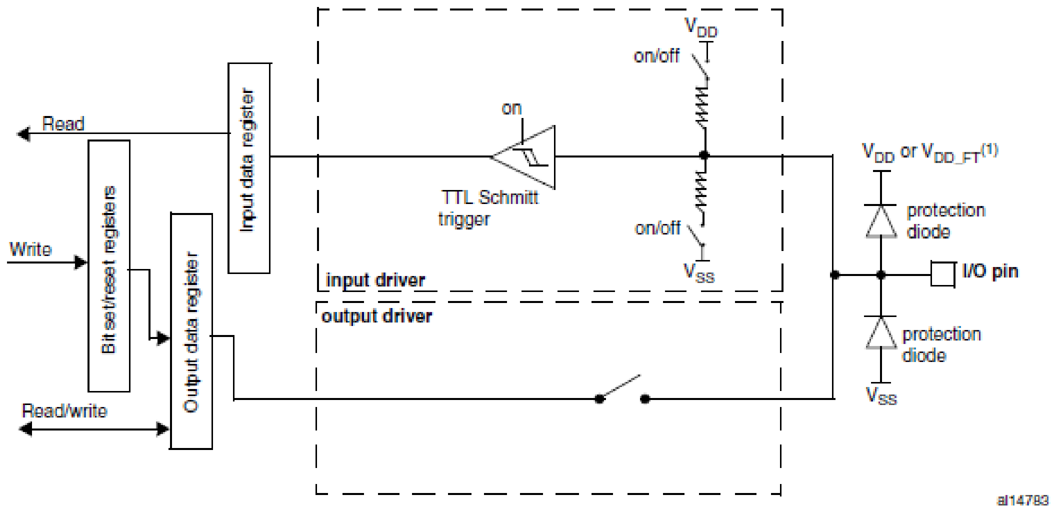


Figure 23 - Input Floating/Pull-Up/Pull-Down Configurations

Since ARM was a different processor, other than certain instruction of how to program, one use of GPIO was to send command in the same syntax. To obtain that, a data structure made of 8 bytes was set up just as the same format of what BS2 sent. After that, this structure was sent on the serial line as a packet. The position could be rapidly calculated and the result will be sent to 18 servos in tiny time, which showed up like they were controlled and updated at the same time.

5.2.6 Timers

The PWM input function of the timer is used to measure the pulse width and the frequency of the PWM wave of the input signals from the sensors.

In this project, channel 1 and channel 2 of timer 2 is used to set up timer 2 as the PWM input mode. The registers of STM32 should be set up as follows.

Follow the guide from the Reference Manual first:

1. Select the active input for TIM2_CCR2: write the CC2S bits to 01 in the TIM2_CCMR2 register (TI2 selected).
2. Select the active polarity for TI1FP2 (used both for capture in TIM2_CCR2 and counter clear): write the CC2P bit to ‘0’ (active on rising edge).
3. Select the active input for TIM2_CCR1: write the CC1S bits to 10 in the TIM2_CCMR2 register (TI2 selected).

4. Select the active polarity for TI1FP1 (used for capture in TIMx_CCR1): write the CC1P bit to ‘1’ (active on falling edge).
5. Select the valid trigger input: write the TS bits to 101 in the TIM2_SMCR register (TI1FP2 selected).
6. Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIM2_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to ‘1’ in the TIM2_CCER register.

After that, as the frequency of the STM32 timer is so fast, 16 bit timer can only measure the pulse less than 0.9 ms. So the prescaler should be used to expand the time and sacrifice the precision which is totally enough.

One GPIO setting is also needed. PA01, which is used as Channel 2 of timer 2, should be set as “floating input” mode.

After all the initial settings above, as soon as we start the timer and the input signal wired to PA01, the registers TIM2_CCR1 and TIM2_CCR2 begin to capture the measurement. TIM2_CCR1 records the pulse width and TIM2_CCR2 records the period. The values of them should be read into variables in a certain frequency so that the change of them could be shown.

5.2.7 USARTs

Universal Synchronous Asynchronous Receiver Transmitter (USART) is another way for serial communication since the pins of USART and those of GPIO are internally connected. The USART offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator. It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication. High-speed data communication is possible by using the DMA for multi-buffer configuration.

The interface is externally connected to another device by three pins. Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its IO port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this IO is used to transmit and receive the data (at USART level, data are then received on SW_RX).

USART in this project was used for communication between ARM and PC. Since before the signal sent as expected, a large amount of failures were gone through.

For example, it could not work out as wanted when testing the code sending position command to PSC. By connecting the PC and STM32 with serial cable through the USART interface, using software called putty, what was being sent to the PSC would be synchronized to show on the PC's screen. In this way, mistakes can be easier to find out.

5.2.8 SPI

The Serial Peripheral Interface (SPI) allows half/full-duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case, it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multi-master configuration.

It may be used for a variety of purposes, including Simplex synchronous transfers on two lines with a possible bidirectional data line or reliable communication using CRC checking.

Usually, the SPI is connected to external devices through four pins:

- MISO: Master In / Slave Out data. This pin can be used to transmit data in slave mode and receive data in master mode.
- MOSI: Master Out / Slave In data. This pin can be used to transmit data in master mode and receive data in slave mode.
- SCK: Serial Clock output for SPI masters and input for SPI slaves.
- NSS: Slave select.

The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

The communication is always initiated by the master. When the master device transmits data to a slave device via the MOSI pin, the slave device responds via the MISO pin. This implies full-duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

Taking configuring the SPI in master mode for example, the configuration procedure is:

1. Select the BR[2:0] bits to define the serial clock baud rate.
2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock.
3. Set the DFF bit to define 8- or 16-bit data frame format.
4. Configure the LSBFIRST bit in the SPI_CR1 register to define the frame format.
5. If the NSS pin is required in input mode, in hardware mode, connect the NSS pin to a high-level signal during the complete byte transmit sequence. In NSS software mode, set the SSM and SSI bits in the SPI_CR1 register. If the NSS pin is required in output mode, the SSOE bit only should be set.
6. The MSTR and SPE bits must be set (they remain set only if the NSS pin is connected to a high-level signal).

In this configuration the MOSI pin is a data output and the MISO pin is a data input.

After the SPI is configured, it's time to function as transmitting and receiving interface. Transmit sequence:

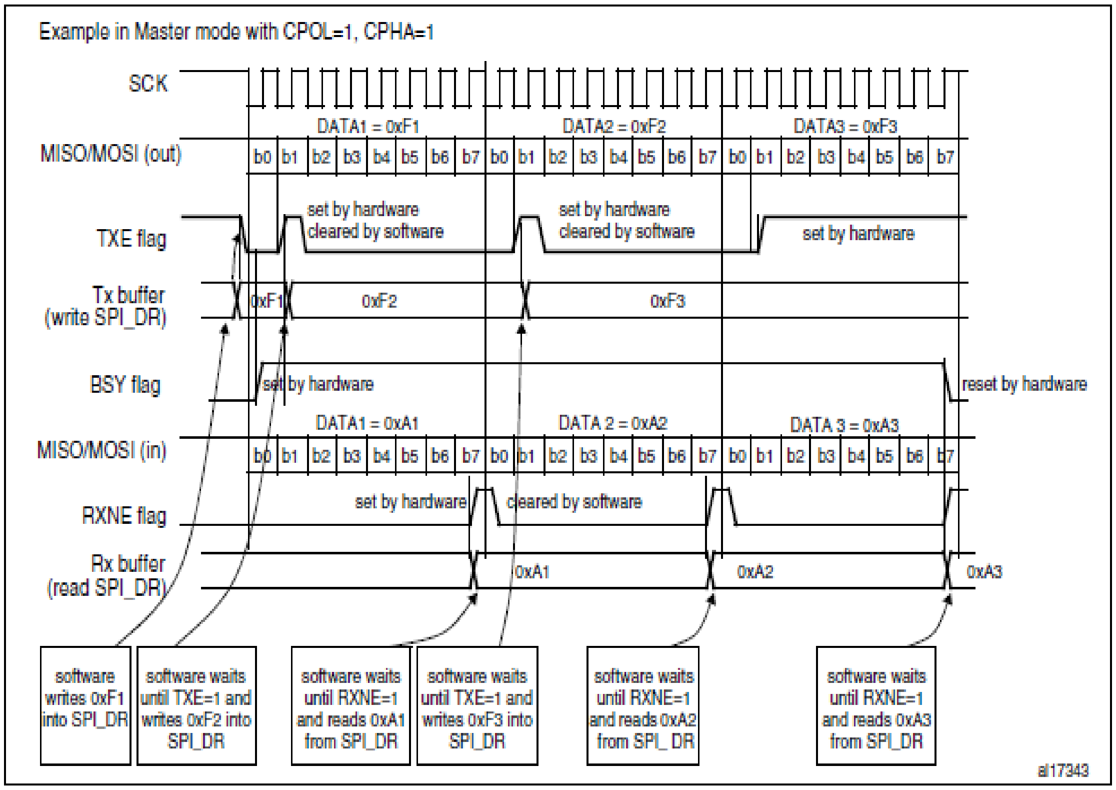
The transmit sequence begins when a byte is written in the Tx Buffer. The data byte is parallel-loaded into the shift register (from the internal bus) during the first bit transmission and then shifted out serially to the MOSI pin MSB first or LSB first depending on the LSBFIRST bit in the SPI_CR1 register. The TXE flag is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

Receive sequence:

For the receiver, when data transfer is complete:

- The data in the shift register is transferred to the RX Buffer and the RXNE flag is set
- An interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register

At the last sampling clock edge the RXNE bit is set; a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value. Clearing the RXNE bit is performed by reading the SPI_DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in the Tx buffer once the transmission is started. Note that TXE flag should be '1 before any attempt to write the Tx buffer is made.



Example in Master mode with CPOL=1, CPHA=1
SCK
MISO/MOSI (out)
DATA1 = 0xF1
DATA2 = 0xF2
DATA3 = 0xF3
TXE flag
set by hardware cleared by software
set by hardware cleared by software
set by hardware
Tx buffer (write SPI_DR)
0xF1
0xF2
0xF3
BSY flag
set by hardware
reset by hardware
MISO/MOSI (in)
DATA1 = 0xA1
DATA 2 = 0xA2
DATA 3 = 0xA3
RXNE flag
set by hardware
cleared by software
Rx buffer (read SPI_DR)
0xA1
0xA2
0xA3
software writes 0xF1 into SPI_DR
software waits until TXE=1 and writes 0xF2 into SPI_DR
software waits until RXNE=1 and reads 0xA1 from SPI_DR
software waits until TXE=1 and writes 0xF3 into SPI_DR
software waits until RXNE=1 and reads 0xA2 from SPI_DR
software waits until RXNE=1 and reads 0xA3 from SPI_DR
ai17343

Figure 24 - TXE/RXNE/BSY Behavior in Master / Full-Duplex Mode

In this project, SPI was supposed to function with ADC chips for ground sensing. Since the current going through the servo was in linear relationship with the servo's torque. That is to say, if the leg touched the ground, to stand still, the current had to rise up. Taking advantage of this relation, MAX 471 chip is chosen to sense the current and turn it into voltage signal. Before the signal can be used, its analogous feature had to be translated into digital one. MCP3208 was an ideal option as an Analog Digital Converter. After digital signal sampled, it became simple to figure out if the feet touched the ground.

Since this method only worked well when the current kept constant, but in the project current corresponded to the pulsed voltage to be a pulsed current as well. It was estimated to take a long time to turn the pulsed current into a constant one with nearly the same effect. Constrained by the time limit of this project, this work can be left to the future work.

5.2.9 HexCrawler 2.0 Code

After completing the background research on Hexapods, it was decided that open source low-level control code should be sought after to make the best use of the allotted time for the project. Team members found a hexapod similar to the one being designed for the MQP project called "Phoenix." Phoenix is a collaborative project headed by lead-programmer Jeroen Janssen. Jeroen's project uses the Phoenix Hexapod Robot Kit for a chassis, and runs custom BASIC code on a Basic Atom Pro 28 microcontroller. SSC 32 servo controllers are used to control 18 servos on the 3-DOF legs. Although the Phoenix Robot was not identical to the robot being designed in this project, it was similar enough that its control code could be used as a basis for the low-level code of HexCrawler 2.0. The code for the project was organized into three main files: HexCrawler.c, which contained the main control loop and kinematics algorithms, HexCrawler_2_cfg.h, which contained the configurations for the wiring and physical dimensions of the robot's chassis, and HexCrawler.h, the header file which included the files with code for the peripherals. The code was developed using µVision 4 IDE made by Keil and run on the STM32F107 development board on the robot.

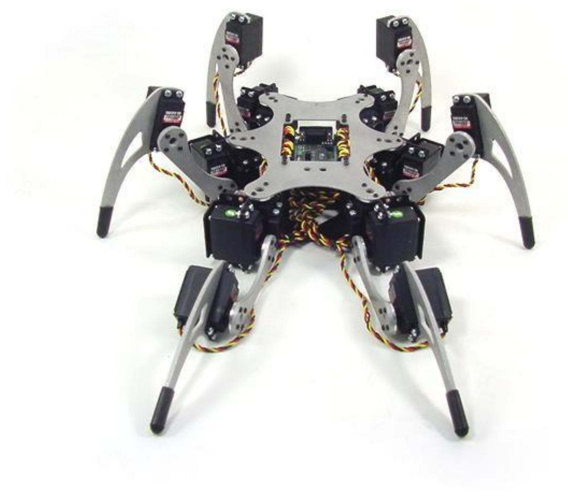


Figure 25 - Phoenix Hexapod Robot

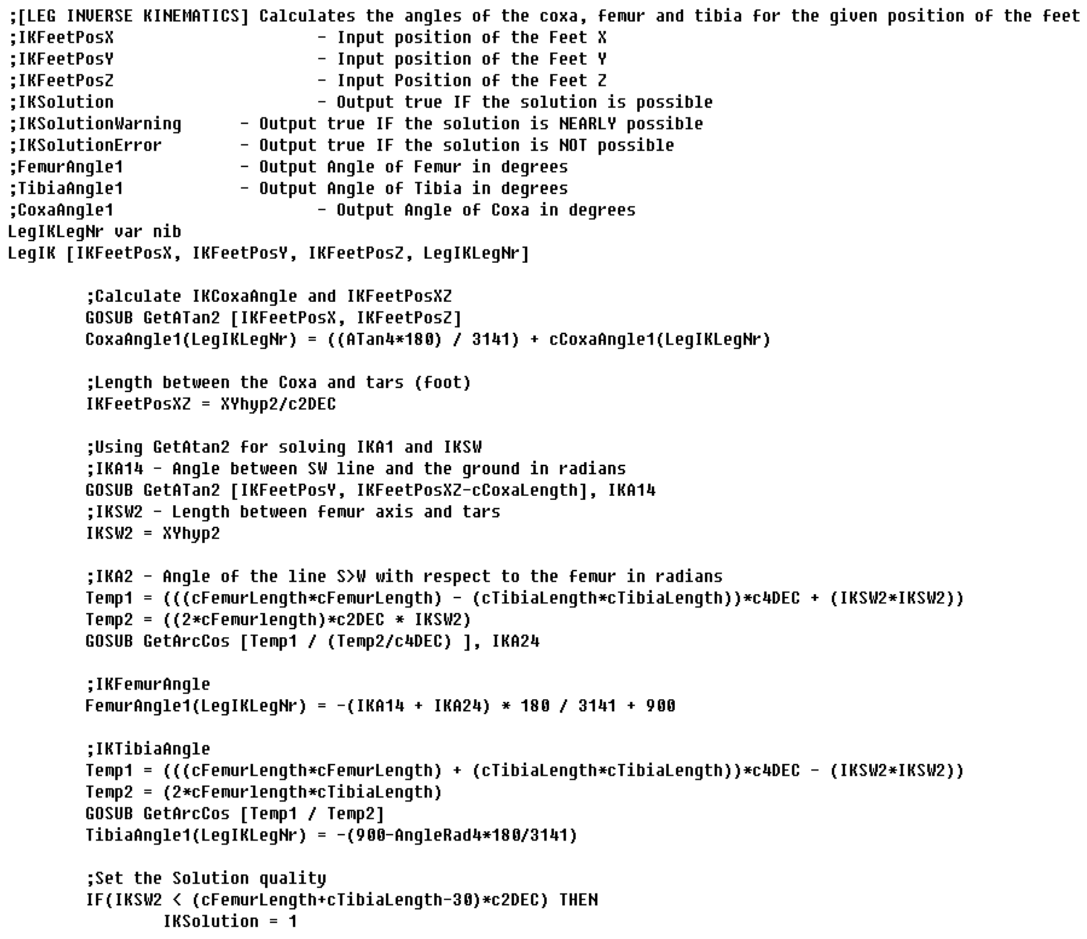
After entering the main() function, the stm32_Init() function was called to configure all of the hardware features of the STM32. This included such options as timers, serial communication, and general-purpose inputs and outputs. Second, a function called InitHexCrawler() was called. This secondary initialization function configured robot specific features such as LEDs, initial leg and body positions, and gait options. After both initialization routines had been completed, the STM32 processor entered and infinite while loop.

In seeing that one of the goals of the project was to implement velocity control so that the robot would move with lower levels of jerk, the first thing done in the control loop was to record the start time. A function called DemoSequence() could optionally be used to test low-level functionality without a connection from a computer. The last thing that executed before the kinematics calculations began was a function that turned the status LEDs on the robot on or off depending on the results of the previous set of kinematics calculations; during the first run, the LEDs would be set to "off."

The kinematics routines of the HexCrawler 2.0 code were executed in five distinct layers, each having a higher level of control than the preceding layer. The lowest level of control was "single leg" control. Using this mode, a user could control any of the six legs individually by inputting an offset in the x, y, and z directions from the foot's original position. This option would be very useful to control a foot outside of its normal gate path, for example, to step onto a staircase.

The second level of leg control took place in the GateSequence() function. This program used several variables, such as gait speed and step size, in order to move the legs in one of eight predefined gait sequences. The operator had a wide range of control over the way the robot walked by changing one or more of the associated gait variables. This function was very useful to accomplish the objective of making HexCrawler 2.0 walk more smoothly than the previous version.

The next function run was associated with a special mode called "Balance." This part of the program was written specifically with the goals of climbing stairs and walking on irregular terrain in mind.



```
;[LEG INVERSE KINEMATICS] Calculates the angles of the coxa, femur and tibia for the given position of the feet
;IKFeetPosX                     - Input position of the Feet X
;IKFeetPosY                     - Input position of the Feet Y
;IKFeetPosZ                     - Input Position of the Feet Z
;IKSolution                     - Output true IF the solution is possible
;IKSolutionWarning      - Output true IF the solution is NEARLY possible
;IKSolutionError        - Output true IF the solution is NOT possible
;FemurAngle1            - Output Angle of Femur in degrees
;TibiaAngle1            - Output Angle of Tibia in degrees
;CoxaAngle1                     - Output Angle of Coxa in degrees
LegIKLegNr var nib
LegIK [IKFeetPosX, IKFeetPosY, IKFeetPosZ, LegIKLegNr]

        ;Calculate IKCoxaAngle and IKFeetPosXZ
        GOSUB GetATan2 [IKFeetPosX, IKFeetPosZ]
        CoxaAngle1(LegIKLegNr) = ((ATan4*180) / 3141) + cCoxaAngle1(LegIKLegNr)

        ;Length between the Coxa and tars (foot)
        IKFeetPosXZ = XYhyp2/c2DEC

        ;Using GetAtan2 for solving IKA1 and IKSW
        ;IKA14 - Angle between SW line and the ground in radians
        GOSUB GetATan2 [IKFeetPosY, IKFeetPosXZ-cCoxaLength], IKA14
        ;IKSW2 - Length between femur axis and tars
        IKSW2 = XYhyp2

        ;IKA2 - Angle of the line S>W with respect to the femur in radians
        Temp1 = (((cFemurLength*cFemurLength) - (cTibiaLength*cTibiaLength))*c4DEC + (IKSW2*IKSW2))
        Temp2 = ((2*cFemurlength)*c2DEC * IKSW2)
        GOSUB GetArcCos [Temp1 / (Temp2/c4DEC) ], IKA24

        ;IKFemurAngle
        FemurAngle1(LegIKLegNr) = -(IKA14 + IKA24) * 180 / 3141 + 900

        ;IKTibiaAngle
        Temp1 = (((cFemurLength*cFemurLength) + (cTibiaLength*cTibiaLength))*c4DEC - (IKSW2*IKSW2))
        Temp2 = (2*cFemurlength*cTibiaLength)
        GOSUB GetArcCos [Temp1 / Temp2]
        TibiaAngle1(LegIKLegNr) = -(900-AngleRad4*180/3141)

        ;Set the Solution quality
        IF(IKSW2 < (cFemurLength+cTibiaLength-30)*c2DEC) THEN
                IKSolution = 1
```

Figure 26 - Phoenix Code Written in BASIC Language

The main purpose of this program was to average the positions of all six feet in three dimensions in order to generate a set of body translation and rotation offsets which would place the robot's body roughly in the middle of its feet. If you were to imagine the HexCrawler beginning to climb a staircase, the BalanceBody() function would begin to tilt the robot's body upward as it placed its front feet on the first stair. By the time the robot was completely on the staircase, its body would be adjusted so that it was parallel to the slope of the staircase.

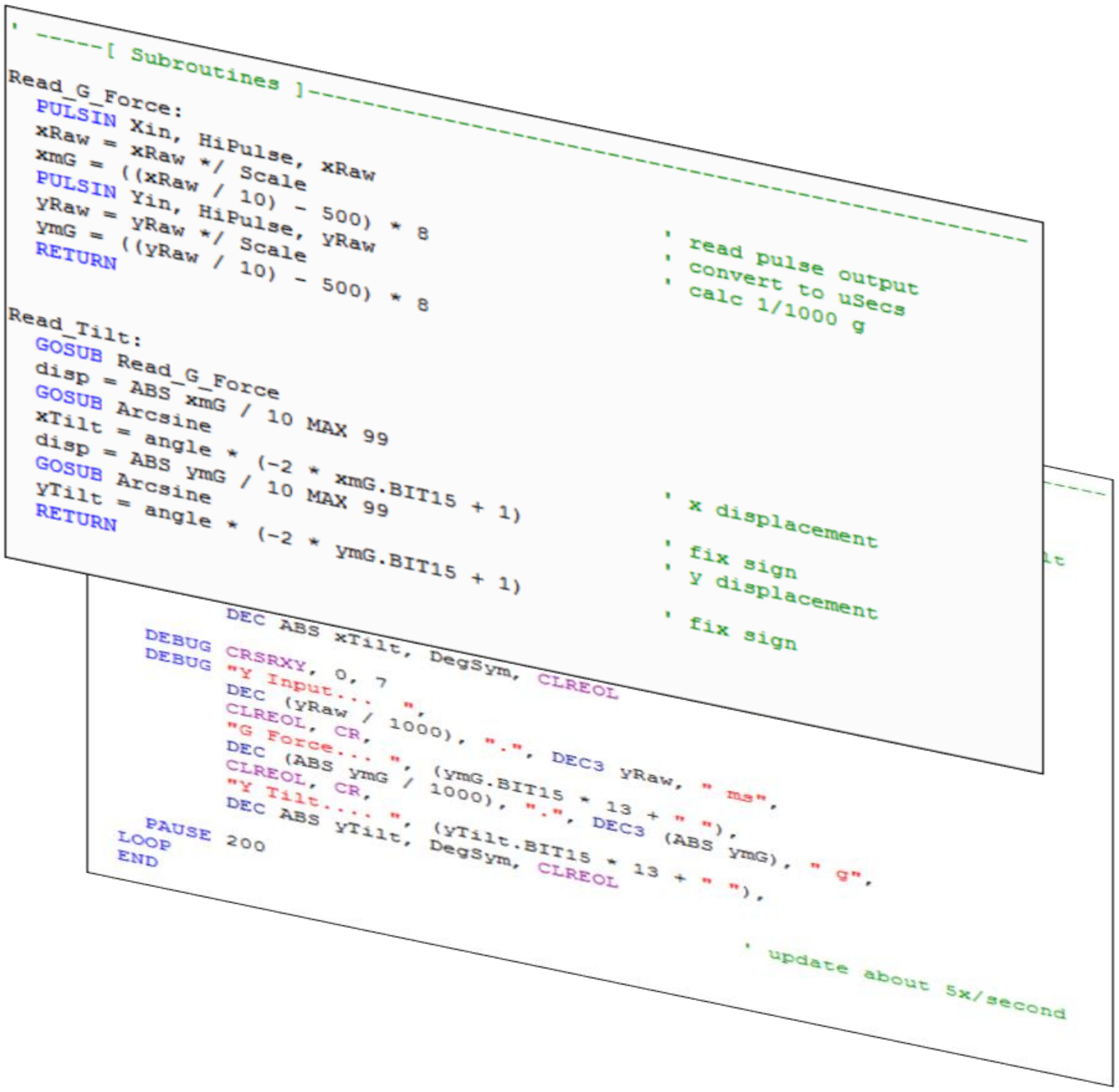
After all of the robot's feet positions were determined and it's optimal body orientation was calculated, the leg and body inverse kinematics functions, LegIK() and BodyIK(), were run for each leg. BodyIK() was the first to run; it would calculate the joint angles required on each leg in order to orient the robot's body as desired. Last, the LegIK() function would calculate the angles required at each of the leg's three joints in order to obtain the desired foot position. Both functions contained complex trigonometric calculations, however, the powerful STM32 microprocessor was able to complete them in mere microseconds.

Once the joint angles had been calculated for each leg, the commands could be sent to the Parallax Propeller Servo Controllers to drive each of the 18 servos. A variable delay was added before communication with the PSC boards began to make sure the previous set of commands had been executed. The length of the delay was calculated using the start time recorded at the beginning of the loop and an end time captured just before the new servo position commands were sent. Communication was handled by the ServoDriver() function. In order to send the commands correctly, the desired servo angles first had to be converted to pulse-width signals that the servos would be able to understand. This was accomplished through the use of a linearization function, AngleToPW(). Eight-byte command packets were sent in groups of three, one to each servo on a given leg, until the position of each of the 18 servos had been specified. Once each leg reached the correct position, the control loop started over with a new set of leg and body position inputs.

5.2.10 Sensors

Accelerometer

In order to make comparison between the old robot and the new one, some parameters had been set, one of which is jerk—acceleration changing speed. Memsic 2125 Dual-Axis Accelerometer (#28017) was chosen to bear this task. Since this sensor was manufactured by Parallax Inc. in cooperation with the BS2, code used on BS2 for this sensor was already written.



```
' -----[ Subroutines ]---------------------------------------------------------

Read_G_Force:
  PULSIN Xin, HiPulse, xRaw                   ' read pulse output
  xRaw = xRaw */ Scale                        ' convert to uSecs
  xmG = ((xRaw / 10) - 500) * 8               ' calc 1/1000 g
  PULSIN Yin, HiPulse, yRaw
  yRaw = yRaw */ Scale
  ymG = ((yRaw / 10) - 500) * 8
  RETURN

Read_Tilt:
  GOSUB Read_G_Force
  disp = ABS xmG / 10 MAX 99                  ' x displacement
  GOSUB Arcsine
  xTilt = angle * (-2 * xmG.BIT15 + 1)        ' fix sign
  disp = ABS ymG / 10 MAX 99                  ' y displacement
  GOSUB Arcsine
  yTilt = angle * (-2 * ymG.BIT15 + 1)        ' fix sign
  RETURN
```

```
            DEC ABS xTilt, DegSym, CLREOL
      DEBUG CRSRXY, 0, 7
      DEBUG "Y Input...  ",
            DEC (yRaw / 1000), ".", DEC3 yRaw, " ms",
            CLREOL, CR,
            "G Force... ", (ymG.BIT15 * 13 + " "),
            DEC (ABS ymG / 1000), ".", DEC3 (ABS ymG), " g",
            CLREOL, CR,
            "Y Tilt.... ", (yTilt.BIT15 * 13 + " "),
            DEC ABS yTilt, DegSym, CLREOL
      PAUSE 200                                ' update about 5x/second
    LOOP
    END
```

Figure 27 - Accelerometer Code Written in BASIC

The Memsic 2125 is a low-cost thermal accelerometer capable of measuring tilt, collision, static and dynamic acceleration, rotation, and vibration with a range of ±3 g on two axes. Memsic provides the 2125 IC in a surface-mount format. Parallax mounts the circuit on a tiny PCB providing all I/O connections so it can easily be inserted on a breadboard or through-hole prototype area.

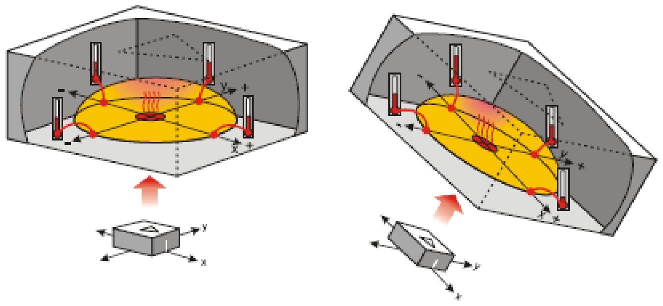
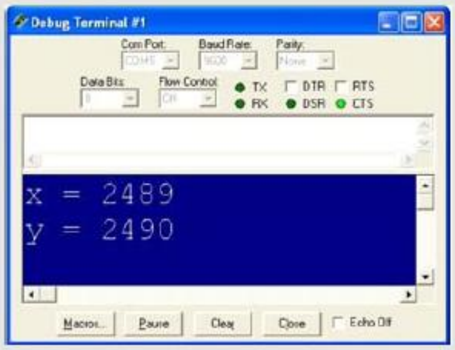


Figure 28 - Memsic Accelerometer Operation

The MX2125 has a chamber of gas with a heating element in the center and four temperature sensors around its edge. When the accelerometer is level, the hot gas pocket rises to the top-center of the chamber, and all the sensors will measure the same temperature.

By tilting the accelerometer, the hot gas will collect closer to some of temperature sensors. By comparing the sensor temperatures, both static acceleration (gravity and tilt) and dynamic acceleration (like taking a ride in a car) can be detected. The MX2125 converts the temperature measurements into signals (pulse durations) that are easy for microcontrollers to measure and decipher.

```
' Smart Sensors and Applications - SimpleTilt.bs2
' Measure room temperature tilt.

'{$STAMP BS2}
'{$PBASIC 2.5}

x               VAR     Word
y               VAR     Word

DEBUG CLS

DO

  PULSIN 6, 1, x
  PULSIN 7, 1, y

  DEBUG HOME, DEC4 ? X, DEC4 ? Y

  PAUSE 100

LOOP
```



```
x = 2489
y = 2490
```

Figure 29 - BASIC Code with Serial Output

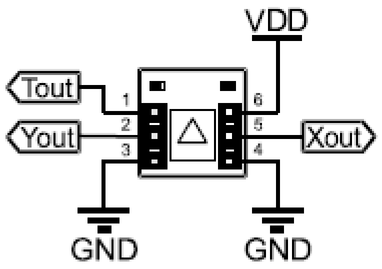


Figure 30 - Accelerometer Pinout

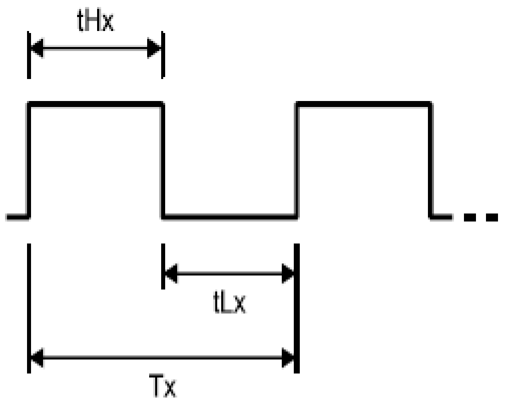


Figure 31 - Accelerometer Duty Cycle Signal

The PULSIN command was a convenient one to measure the pulse width but only usable in BS2. When it came to the ARM processor, command with this function had to be created from nothing. GPIO and TIM were used to realize this command.

Certain GPIO pins were connected to the Yout and Xout and then the work is to test the width of high voltage and pulse circle. TIM interrupt and register were called measuring. If the voltage on the pins jumped to high, TIM interrupt was triggered with a register starting to record time. When the voltage on the pins jumped low, another interrupt was generated with the register stopping recording the time. As a result, tHx was measured. Appling the same method, Tx can be measured. To obtain this, it was just needed to change values of certain registers and then read the data of time.

Infrared Distance Sensor

Infrared sensor was one kind of distance or obstacle sensor, which enabled the robot to set an "image" of the world it stood in. Environment sensing was essential for a robot if it was expected to be more automatic and intelligent.

Sharp GP2D12 Analog Distance Sensor (#605-00003) was the one used in this project. The Sharp GP2D12 is an analog distance sensor that uses infrared to detect an object between 10 cm and 80 cm away. The GP2D12 provides a non-linear voltage output in relation to the distance an object is from the sensor and interfaces easily using any analog to digital converter.

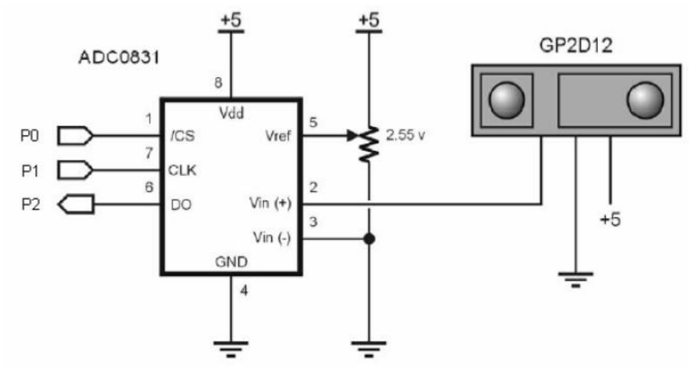


Figure 32 - Infrared Sensor Configuration

The GP2D12 was connected to ADC0831 as shown in the circuit. The potentiometer connected to the Vref pin on the ADC0831 was being used as a voltage divider to set the reference voltage to 2.55 volts. On the ADC0831 this would give a voltage of 0 to 2.55 volts. This gave us a resolution of 0.01 volts per step from the ADC.

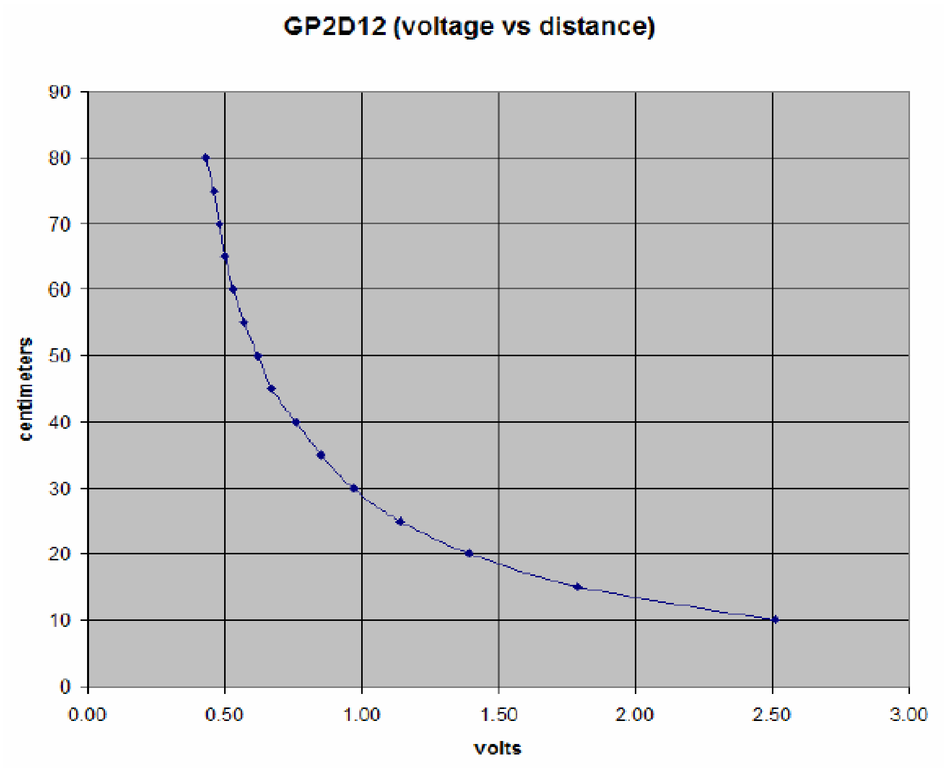


Figure 33 - Relationship between distance and voltage of GP2D12

As shown above, because the output of the GP2D12 was not linear, a way was needed to be found to determine what distances correspond to what voltages. One way of calibrating the sensor was to measure the voltage output of the GP2D12 at given fixed distances, in centimeters, as shown in the chart below. Once this information was founded, these numbers could be stored in the EEPROM DATA for the program. This sensor can be used in short-distance field.

Ultrasonic Distance Sensor

Ultrasonic distance sensor was the other sensor expected to be used in this project.

The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to microcontrollers such as the BASIC Stamp 2, requiring only one I/O pin.

The PING))) sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width, the distance to target can easily be calculated.

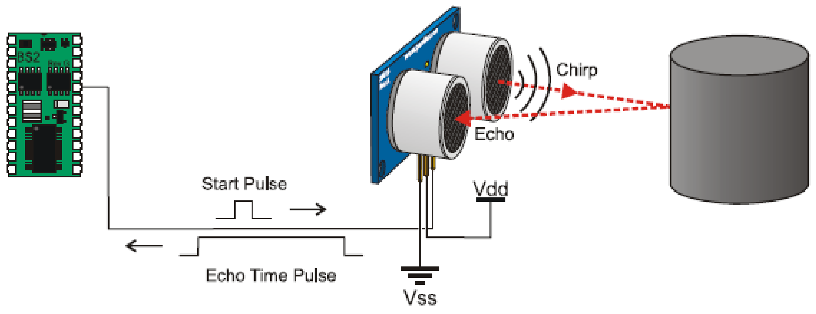


Figure 34 - Ultrasonic Sensor Operation

The PING))) sensor detects objects by emitting a short ultrasonic burst and then "listening" for the echo. Under control of a host microcontroller (trigger pulse), the sensor emits a short 40 kHz (ultrasonic) burst.

This burst travels through the air, hits an object and then bounces back to the sensor. The PING))) sensor provides an output pulse to the host that will terminate when the echo is detected; hence the width of this pulse corresponds to the distance to the target.

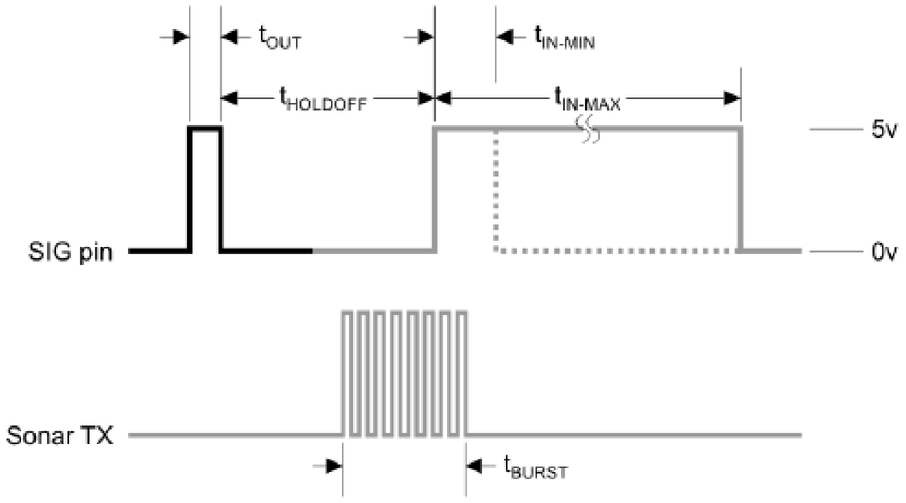


Figure 35 - Ultrasonic Sensor Communication

Chapter 6: Conclusions

The goal of this project was to improve the functionality of the DEPUSH HexCrawler robot to better meet the changing demands of the higher education science, technology, engineering and mathematics (STEM) market. Upon meeting with representatives from DEPUSH, it was learned that this project would be left relatively wide open with few concrete design requirements. Their objective behind this reasoning was to allow our group creative freedom to pursue a number of possible ideas and alternatives for an improved robot system without being constrained to existing ideas and technologies.

We began the project by first brainstorming and identifying new ideas and concepts for delivering an improved six-legged walking robot system, including completely redesigned legs and chassis, stronger servo-motors, more environmental sensors, more powerful embedded control system, wireless communications, and remote control of the robot. The improved legs for the new robot were incorporated a third degree of freedom, used fully symmetrical parts, and were designed in SolidWorks. The legs and the new chassis were all designed to use 1.6mm 5052-alloy aluminum sheet metal with a black anodized or powder-coated protective finish. After the design was completed, complete manufacturing drawings were created and sent to an offsite machine shop for fabrication. To eliminate complexity from the previous design, all linkages were eliminated and each mechanical joint on the robot was directly powered by face mounting the aluminum structural components onto the servo horn. To mechanically power the new robot, Hitec HS-645MG servos were chosen for all eighteen revolute joints due to their peak torque per cost ratio and availability from the sponsor company.

To handle the increased processor workload from the addition of a third degree of freedom per leg, a much faster, C-based Cortex-M3 STM32F10x family of embedded controller processors were chosen, with two specific models chosen for further development. To convert the signal from the embedded controller into a signal usable by the HS-645MG servos, dual 16-channel serial servo controller daughter boards were utilized. Software control of the robot was accomplished by porting the open-source *Phoenix* hexapod inverse-kinematic software library into C to run on the STM32F10x processors. This software library allowed fine control of individual legs, walking in different gaits, and full, six-degree of freedom offset capabilities for the chassis while doing the prior two objectives.

In order to achieve distributed control of the robot to allow for more sophisticated teleported, semi-autonomous, or fully autonomous operation of the new robot, it was determined that distributed computing between a computer and the robot's embedded controller was necessary. After looking through available technologies, it was decided to implement the communication protocol using TCP/IP networking due to its flexibility and nearly complete market penetration in all modern computing platforms. In this model, a lightweight HTTP server would be installed on the robot's embedded controller, along with all low-level control code. Sensor and internal state readings would be read from the embedded controller, processed on a high-level computing device, culminating with high-level commands for desired actions being sent to the embedded controller for execution. The high-level control code and tele-operation control would be handled by a graphical Java application running on a personal computer.

While our group was able to complete successfully the tasks outlined in the first two of the three preceding paragraphs, the tasks outlined in the third remain a work in progress. Delays in the procurement of components, manufacturing turnaround time, technical manual language barriers, return-to-manufacturing of defective parts, and lack of resources hampered progress on our ambitious goals. Despite these obstacles, significant progress was made and our final robot represented a considerable improvement over the original HexCrawler robot. With this in mind, the academic intent of this project was fulfilled, and many complex, real-world, engineering, project management and international collaboration skills were acquired.

Chapter 7: Future Work Recommendations

7.1 Mechanical Subsystem

7.1.1 Dynamic Modeling and Analysis

Since the improvement of this project was compressed into an eight-week time span, the amount of modeling and analyzing of the new design was limited by the manufacturing turnaround and software development times. Thus, the structural components of the robot were subjected to static loading tests in SolidWorks, and static torque analysis was performed for each of the three joints of the robot's legs. Further stress and fatigue tests could be carried out to determine the useful lifespan of the components. Where doubts existed, the design was erred to the over-designed side to ensure that it would perform without failing. With additional time, a full static and dynamic analysis of the robot should be performed to continue optimization its design.

7.1.2 Payload Capability

To allow the robot to carry heavier loads, such as auxiliary battery packs, higher power computation platforms such as a netbook, or other features not yet thought of, additional sheet metal structural parts may be necessary to mount securely these onto the main robot chassis. This task is facilitated by the modular mounting slots designed into the new robot chassis in both metric and imperial units, to allow a wide variety of components to be added to the robot.

7.2 Electrical and Power Subsystem

7.2.1 Batteries

Due to budget limitations, improved battery packs were not purchased for this project, as they were not necessary to successfully test the new robot controls. To avoid being tethered to power supplies, new robot batteries will be required to power the robot. However, the improved Hitec HS-645MG servos present a challenge for proper battery selection; each of these servos can draw as much as 1.8 amps at stall condition. While our ten amp power supply was more than sufficient for testing walking code (with the average current draw being around five amps at six volts), the torque necessary to lift the robot "onto its feet" from a sitting position was beyond this capacity. Thus, a battery technology will need a high peak current flow in addition to a high capacity to allow for extended operation. NiMH and Lithium-based batteries can achieve these results.

7.2.2 Fuse and System Wiring

Protection of the electronics system on the current robot is achieved using the internal circuitry of the power supply. However, with a switch to a mobile, battery-based power system, the robot will need protection from short circuits and other undesirable electrical conditions. This will require the addition of either fuses or resettable circuit breakers to prevent short circuits from destroying the robot’s batteries, battery sensors to monitor voltage and temperature, and protections against electrostatic shocks for critical electronic components.

7.3 Sensor and Feedback Subsystem

7.3.1 Motor Position/Terrain Sensing

To implement a terrain adapting walking algorithm, closed loop feedback of the current position of each joint in the robot’s legs and detection of ground contact will be necessary. By sensing the ground, the inverse-kinematics control can then be changed with an offset for the desired location of each leg based sensed detection of the ground. This will enable the robot leg to stop lifting the robot up when it hits the ground, allowing the robot to conform to rough terrain.

7.3.2 Obstacle Sensing

As a corollary to the previous recommendation, sensors to accurate sense obstacles in the path of the HexCrawler will be necessary for more advanced operation. This will allow the robot to determine whether the size of an obstacle (in the form of an object like a rock, or the absence of an object, such as a hole) is too large to allow the robot to conquer it.

7.4 Embedded Control Subsystem

7.4.1 Interface Management

Since the improved robot was designed to be controlled using different embedded controllers in the Cortex-M3 STM32F10x family, standard interfaces and documentation must be necessary to allow new or replacement components to communicate with the rest of the system. This can be accomplished by writing different configuration header files for each of the desired microcontrollers. An end user would simply have to specify which configuration they are using in the code to begin working with the HexCrawler system.

7.5 Low-Level Robot Firmware

The low-level code running on the robot was mostly complete at the end of the project. The multi-layered inverse kinematics algorithms worked well to control the body's six degrees of freedom.

Unfortunately, because the high-level control was never fully implemented, the planned terrain handling and stair climbing capabilities of the robot were abandoned. Should a team wish to realize these features, ground detection would definitely need to be implemented. Our team had a plan to use current sensing to accomplish this task, and even had the hardware required to do so, just not enough time. Once a high-level control system is fully implemented, the low-level code, especially some of the advanced velocity features, would have to be experimented with and fine-tuned.

7.6 Wireless Communications

7.6.1 Wireless Operation

While the robot (when controlled by the STM32F107VC embedded controller board) can communicate in theory wirelessly, it lacks the hardware to do so. Testing of communication on the improved robot was accomplished via a crossover Ethernet cable with the device connected directly to the Ethernet port on a personal computer. While this is a desirable condition in places were wireless operation is unreliable due to high signal noise or heavily regulated or outright banned due to school or corporate policies, the addition of wireless communication would benefit a significant majority of users.

To add wireless networking capability to the robot would require a wireless router and a non-crossover Ethernet cable. By connecting the embedded controller Ethernet port to one of the output ports of the wireless router, the HTTP server running on the embedded controller would negotiate a DHCP IP address to which all robot communication would be channeled. On most routers, this default IPv4 address is 192.168.1.100 (if no other devices are connected to the network).

7.6.2 Alternative Technologies

Besides wireless 802.11 technologies, alternatives (sans HTTP server) are available. ZigBee and Bluetooth are two commercial radio-frequency communication technologies that can be adapted for use on the new robot. These technologies can be implemented as an alternative to the 802.11 wireless networking.

7.7 Computer Control Application

7.7.1 Java GUI Application

Due to time constraints, there remains a lot of work to be completed on the Java GUI application. Most of the internal, non-GUI functionality needs to be implemented and tested. In addition, while an initial interface was created, testing and feedback from testing mockups of the software could provide valuable advice on how to improve the user experience.

7.7.2 Mobile Device Application

Due to the proliferation of mobile computing platforms, such as iOS, Android and Windows Mobile, the development of an application specifically designed for mobile devices could prove to be very valuable asset for remotely controlling the robot without the need for additional hardware. Communication with the mobile device would be accomplished via 802.11 wireless networking, which is present in nearly all smartphones and mobile computers.

Works Cited

DEPUSH Technology. (2010). *DEPUSH Educational Products (translated from Chinese)*. Retrieved July 2010, from DEPUSH.com: http://www.depush.com/

Mingyan, M. (2010, July). Distributed Computing. (S. Yuwen, Interviewer)

深圳市德普施科技有限公司. (2010). *深圳市德普施科技有限公司 (Depp Shi Technology Co)*. Retrieved August 2010, from 勤加缘 (qjy168.com): http://www.qjy168.com/shop/index_151555.html

"BASIC Stamp General Information." *Parallax Home*. Web. 20 Aug. 2010. <http://www.parallax.com/tabid/214/Default.aspx>.

"China Building Code." *Sustainable Urban Housing in China*. Web. 20 Aug. 2010. <http://chinahousing.mit.edu/english/resources/BuildingCode.html>.

"CrustCrawler - 3DOF HexCrawler Hexapod." *CrustCrawler Robotics - Robotics Kits, Robotic Arms,ROV/AUV Thrusters, Bioloids, Servos*. Web. 20 Aug. 2010. <http://www.crustcrawler.com/products/hexcrawler_hdats.php?prod=4>.

"CrustCrawler - HexCrawler (Hexapod Robot)." *CrustCrawler Robotics - Robotics Kits, Robotic Arms,ROV/AUV Thrusters, Bioloids, Servos*. Web. 20 Aug. 2010. <http://www.crustcrawler.com/products/hexcrawler.php?prod=1>.

"Hexapod Robot V4 - Walking with Terrain Adaptation - Micromagic Systems." *Google Videos*. Web. 20 Aug. 2010. <http://video.google.com/videoplay?docid=-2642846886265937727#docid=5260960445756262526>.

"Hexapod Robot V4 - Walking with Terrain Adaptation - Micromagic Systems." *Google Videos*. Web. 20 Aug. 2010. <http://video.google.com/videoplay?docid=-2642846886265937727#>.

"Home Tips : Building Codes for Stairs & Staircases." *Home Tips - The Web's Most Helpful Home Improvement and Repair Site*. Web. 20 Aug. 2010. <http://www.hometips.com/how-it-works/stairs-staircases-building-codes.html>.

"IBC Stairs Code." *Modular Industrial Offices, Mezzanine Floors, Steel Guard Bollards, Material Lifts, Safety Gates*. Web. 20 Aug. 2010. <http://www.amezz.com/ibc-stairs-code.htm>.

"Lynxmotion - Phoenix." *Lynxmotion Robot Kits*. Web. 20 Aug. 2010. <http://www.lynxmotion.com/c-117-phoenix.aspx>.

"YouTube - B.F. Hexapod - Terrain Adaptation - Test 01." *YouTube - Broadcast Yourself.* Web. 20 Aug. 2010. <http://www.youtube.com/watch?v=3DkL6BQiVKQ>.

"YouTube - Hexcrawler Spinning in Place." *YouTube - Broadcast Yourself.* Web. 20 Aug. 2010. <http://www.youtube.com/watch?v=1RRMi6oMAdk>.

“Hexapod Photos”, *Baidu Image Search*, 2010 Aug, <hexapodrobot.com, robotshop.com, lynxmotion.net, technono.com>

“Servo Information”, <http://www.hitecrcd.com>

"ARM Reference Manual RM0008" from the website of STMiccontroller

"Analysis of Outline Features of Morphology Design for Modern Machine Tools", LIU Xiaopeng, ZHENG Xinjian, ZHANG Wenqiao. School of Mechanical Engineering, Hubei Univ. of Technology, Wuhan, China. Wuhan Heavy Duty Machine Tool Group co., Ltd.

"Propeller Servo Controller USB (#28830)" from websites of Parallax Inc.

"Parallax Servo Controller - USB (#28823) Rev B (16-Channel Servo Control with a USB Interface)" from websites of Parallax Inc.

Phoenix

"Memsic 2125 Dual-Axis Accelerometer (#28017)" from websites of Parallax Inc.

"Sharp GP2D12 Analog Distance Sensor (#605-00003)" from websites of Parallax Inc.

"PING)))™ Ultrasonic Distance Sensor (#28015)" from websites of Parallax Inc.

"MCP3204/3208 2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface" from websites of Microchip Corp.

Appendix A: HexCrawler 2.0 Code

```
//Project Lynxmotion Phoenix
//Description: Phoenix software
//Software version: V2.0
//Date: 29-10-2009
//Programmer: Jeroen Janssen (aka Xan)
//Modified by: Daniel Earley
//
//Hardware setup: ARM STM32F107, Parallax Servo Controllers
//
//NEW IN V2.0
//    - Moved to fixed point calculations
//    - Inverted BodyRotX and BodyRotZ direction
//    - Added deadzone for switching gaits
//    - Added GP Player
//    - SSC version check to enable/disable GP player
//    - Controls changed, Check contol file for more information
//    - Added separate files for control and configuration functions
//    - Solved bug at turn-off sequence
//    - Solved bug about legs beeing lift at small travelvalues in 4 steps
tripod gait
//    - Solved bug about body translate results in rotate when balance is on
(Kåre)
//    - Sequence for wave gait changed (Kåre)
//    - Improved ATan2 function for IK (Kåre)
//    - Added option to turn on/off eyes (leds)
//    - Moving legs to init position improved
//    - Using Indexed values for legs
//    - Added single leg control
//
//KNOWN BUGS:
//    - None at the moment ;)
//
//Project file order:
//    1. Phoenix_cfg.bas
//    2. Phoenix_V2x.bas
//    3. Phoenix_Control_xxx.bas

//====================================================================
//[INCLUDES]
#include "HexCrawler.h"


//--------------------------------------------------------------------
//[TABLES]
//ArcCosinus Table
//Table build in to 3 part to get higher accuracy near cos = 1.
//The biggest error is near cos = 1 and has a biggest value of 3*0.012098rad
= 0.521 deg.
//-   Cos 0 to 0.9 is done by steps of 0.0079 rad. (1/127)
//-   Cos 0.9 to 0.99 is done by steps of 0.0008 rad (0.1/127)
//-   Cos 0.99 to 1 is done by step of 0.0002 rad (0.01/64)
//Since the tables are overlapping the full range of 127+127+64 is not
necessary. Total bytes: 277
BYTE GetACos[278] =
{255,254,252,251,250,249,247,246,245,243,242,241,240,238,237,236,234,233,232,
231,229,228,227,225,

      224,223,221,220,219,217,216,215,214,212,211,210,208,207,206,204,203,201
,200,199,197,196,195,193,

      192,190,189,188,186,185,183,182,181,179,178,176,175,173,172,170,169,167
,166,164,163,161,160,158,

      157,155,154,152,150,149,147,146,144,142,141,139,137,135,134,132,130,128
,127,125,123,121,119,117,

      115,113,111,109,107,105,103,101,98,96,94,92,89,87,84,81,79,76,73,73,73,
72,72,72,71,71,71,70,70,

      70,70,69,69,69,68,68,68,67,67,67,66,66,66,65,65,65,64,64,64,63,63,63,62
,62,62,61,61,61,60,60,59,

      59,59,58,58,58,57,57,57,56,56,55,55,55,54,54,53,53,53,52,52,51,51,51,50
,50,49,49,48,48,47,47,47,

      46,46,45,45,44,44,43,43,42,42,41,41,40,40,39,39,38,37,37,36,36,35,34,34
,33,33,32,31,31,30,29,28,

      28,27,26,25,24,23,23,23,23,22,22,22,22,21,21,21,21,20,20,20,19,19,19,19
,18,18,18,17,17,17,17,16,

      16,16,15,15,15,14,14,13,13,13,12,12,11,11,10,10,9,9,8,7,6,6,5,3,0};

//Sin table 90 deg, persision 0.5 deg (180 values)
WORD GetSin[181] = {0, 87, 174, 261, 348, 436, 523, 610, 697, 784, 871, 958,
1045, 1132, 1218, 1305, 1391, 1478, 1564,
                        1650, 1736, 1822, 1908, 1993, 2079, 2164, 2249, 2334,
2419, 2503, 2588, 2672, 2756, 2840, 2923, 3007,
                        3090, 3173, 3255, 3338, 3420, 3502, 3583, 3665, 3746,
3826, 3907, 3987, 4067, 4146, 4226, 4305, 4383,
                        4461, 4539, 4617, 4694, 4771, 4848, 4924, 4999, 5075,
5150, 5224, 5299, 5372, 5446, 5519, 5591, 5664,
                        5735, 5807, 5877, 5948, 6018, 6087, 6156, 6225, 6293,
6360, 6427, 6494, 6560, 6626, 6691, 6755, 6819,
                        6883, 6946, 7009, 7071, 7132, 7193, 7253, 7313, 7372,
7431, 7489, 7547, 7604, 7660, 7716, 7771, 7826,
                        7880, 7933, 7986, 8038, 8090, 8141, 8191, 8241, 8290,
8338, 8386, 8433, 8480, 8526, 8571, 8616, 8660,
                        8703, 8746, 8788, 8829, 8870, 8910, 8949, 8987, 9025,
9063, 9099, 9135, 9170, 9205, 9238, 9271, 9304,
                        9335, 9366, 9396, 9426, 9455, 9483, 9510, 9537, 9563,
9588, 9612, 9636, 9659, 9681, 9702, 9723, 9743,
                        9762, 9781, 9799, 9816, 9832, 9848, 9862, 9876, 9890,
9902, 9914, 9925, 9935, 9945, 9953, 9961, 9969,
                         9975, 9981, 9986, 9990, 9993, 9996, 9998, 9999,
10000};
```

```
//Build tables for Leg configuration like I/O and MIN/MAX values to easy
access values using a FOR loop
//Constants are still defined as single values in the cfg file to make it
easy to read/configure

//SSC Pin numbers
BYTE cCoxaPin[6] = {cRRCoxaPin,  cRMCoxaPin,  cRFCoxaPin,  cLRCoxaPin,
cLMCoxaPin,  cLFCoxaPin};
BYTE cFemurPin[6] = {cRRFemurPin, cRMFemurPin, cRFFemurPin, cLRFemurPin,
cLMFemurPin, cLFFemurPin};
BYTE cTibiaPin[6] = {cRRTibiaPin, cRMTibiaPin, cRFTibiaPin, cLRTibiaPin,
cLMTibiaPin, cLFTibiaPin};

//Servo Center Offsets (ms)
WORD cCoxaMiddle[6] = {cRRCoxaMiddle, cRMCoxaMiddle, cRFCoxaMiddle,
cLRCoxaMiddle, cLMCoxaMiddle, cLFCoxaMiddle};
WORD cFemurMiddle[6] = {cRRFemurMiddle, cRMFemurMiddle, cRFFemurMiddle,
cLRFemurMiddle, cLMFemurMiddle, cLFFemurMiddle};
WORD cTibiaMiddle[6] = {cRRTibiaMiddle, cRMTibiaMiddle, cRFTibiaMiddle,
cLRTibiaMiddle, cLMTibiaMiddle, cLFTibiaMiddle};

//Min / Max values
SHORT cCoxaMin1[6] = {cRRCoxaMin1,  cRMCoxaMin1,  cRFCoxaMin1,  cLRCoxaMin1,
cLMCoxaMin1,  cLFCoxaMin1};
SHORT cCoxaMax1[6] = {cRRCoxaMax1,  cRMCoxaMax1,  cRFCoxaMax1,  cLRCoxaMax1,
cLMCoxaMax1,  cLFCoxaMax1};
SHORT cFemurMin1[6] = {cRRFemurMin1, cRMFemurMin1, cRFFemurMin1, cLRFemurMin1,
cLMFemurMin1, cLFFemurMin1};
SHORT cFemurMax1[6] = {cRRFemurMax1, cRMFemurMax1, cRFFemurMax1, cLRFemurMax1,
cLMFemurMax1, cLFFemurMax1};
SHORT cTibiaMin1[6] = {cRRTibiaMin1, cRMTibiaMin1, cRFTibiaMin1, cLRTibiaMin1,
cLMTibiaMin1, cLFTibiaMin1};
SHORT cTibiaMax1[6] = {cRRTibiaMax1, cRMTibiaMax1, cRFTibiaMax1, cLRTibiaMax1,
cLMTibiaMax1, cLFTibiaMax1};

//Body Offsets (distance between the center of the body and the center of the
coxa)
CHAR cOffsetX[6] = {cRROffsetX, cRMOffsetX, cRFOffsetX, cLROffsetX,
cLMOffsetX, cLFOffsetX};
CHAR cOffsetZ[6] = {cRROffsetZ, cRMOffsetZ, cRFOffsetZ, cLROffsetZ,
cLMOffsetZ, cLFOffsetZ};

//Default leg angle
SHORT cCoxaAngle[6] = {cRRCoxaAngle1, cRMCoxaAngle1, cRFCoxaAngle1,
cLRCoxaAngle1, cLMCoxaAngle1, cLFCoxaAngle1};

//Start positions for the leg
SHORT cInitPosX[6] = {cRRInitPosX, cRMInitPosX, cRFInitPosX, cLRInitPosX,
cLMInitPosX, cLFInitPosX};
SHORT cInitPosY[6] = {cRRInitPosY, cRMInitPosY, cRFInitPosY, cLRInitPosY,
cLMInitPosY, cLFInitPosY};
SHORT cInitPosZ[6] = {cRRInitPosZ, cRMInitPosZ, cRFInitPosZ, cLRInitPosZ,
cLMInitPosZ, cLFInitPosZ};
//--------------------------------------------------------------------
//[REMOTE]
```

```
#define cTravelDeadZone  4    //The deadzone for the analog input from the
remote
//====================================================================
//[ANGLES]
SHORT CoxaAngle1[6];    //Actual Angle of the horizontal hip, decimals = 1
SHORT FemurAngle1[6];   //Actual Angle of the vertical hip, decimals = 1
SHORT TibiaAngle1[6];   //Actual Angle of the knee, decimals = 1
//--------------------------------------------------------------------
//[POSITIONS SINGLE LEG CONTROL]
BYTE SLHold;                  //Single leg control mode

SHORT LegPosX[6]; //Actual X Posion of the Leg
SHORT LegPosY[6]; //Actual Y Posion of the Leg
SHORT LegPosZ[6]; //Actual Z Posion of the Leg
//--------------------------------------------------------------------
//[INPUTS]
BYTE butA;
BYTE butB;
BYTE butC;

BYTE prev_butA;
BYTE prev_butB;
BYTE prev_butC;
//--------------------------------------------------------------------
//[GP PLAYER]
BYTE GPStart;                 //Start the GP Player
BYTE GPSeq;                   //Number of the sequence
BYTE GPVerData[3];            //Received data to check the SSC Version
BYTE GPEnable;                //Enables the GP player when the SSC version
ends with "GP<cr>"
//--------------------------------------------------------------------
//[OUTPUTS]
BYTE LedA;  //Red
BYTE LedB;  //Green
BYTE LedC;  //Orange
BYTE Eyes;  //Eyes output
//--------------------------------------------------------------------
//[SERVO SERIAL PACKET]
SERVO_PACKET pac;

//--------------------------------------------------------------------
//[USART Initialization Struct]
//USART_InitTypeDef Usart2Init;

//--------------------------------------------------------------------
//[USART Clock Initialization Struct]
//USART_ClockInitTypeDef Usart2ClockInit;

//--------------------------------------------------------------------
//[VARIABLES]
BYTE Index;             //Index universal used
BYTE LegIndex;          //Index used for leg Index Number

//GetSinCos / ArcCos
SHORT AngleDeg1;        //Input Angle in degrees, decimals = 1
WORD ABSAngleDeg1;            //Absolute value of the Angle in Degrees,
decimals = 1
```

```
SHORT sin4;                    //Output Sinus of the given Angle, decimals = 4
SHORT cos4;                    //Output Cosinus of the given Angle, decimals =
4
SHORT AngleRad4;         //Output Angle in radials, decimals = 4
BYTE NegativeValue;            //If the the value is Negative

//GetAtan2
SHORT AtanX;             //Input X
SHORT AtanY;             //Input Y
SHORT Atan4;             //ArcTan2 output
SHORT XYhyp2;            //Output presenting Hypotenuse of X and Y

//Body position
CHAR BodyPosX;           //Global Input for the position of the body
SHORT BodyPosY;
CHAR BodyPosZ;

//Body Inverse Kinematics
CHAR BodyRotX;           //Global Input pitch of the body
CHAR BodyRotY;           //Global Input rotation of the body
CHAR BodyRotZ;           //Global Input roll of the body
SHORT PosX;              //Input position of the feet X
SHORT PosZ;              //Input position of the feet Z
SHORT PosY;              //Input position of the feet Y
CHAR RotationY;          //Input for rotation of a single feet for the gait
SHORT sinA4;             //Sin buffer for BodyRotX calculations
SHORT cosA4;             //Cos buffer for BodyRotX calculations
SHORT sinB4;             //Sin buffer for BodyRotX calculations
SHORT cosB4;             //Cos buffer for BodyRotX calculations
SHORT sinG4;             //Sin buffer for BodyRotZ calculations
SHORT cosG4;             //Cos buffer for BodyRotZ calculations
SHORT TotalX;            //Total X distance between the center of the body and
the feet
SHORT TotalZ;            //Total Z distance between the center of the body and
the feet
SHORT BodyIKPosX; //Output Position X of feet with Rotation
SHORT BodyIKPosY; //Output Position Y of feet with Rotation
SHORT BodyIKPosZ; //Output Position Z of feet with Rotation

//Leg Inverse Kinematics
SHORT IKFeetPosX; //Input position of the Feet X
SHORT IKFeetPosY; //Input position of the Feet Y
SHORT IKFeetPosZ; //Input Position of the Feet Z
SHORT IKFeetPosXZ;       //Diagonal direction from Input X and Z
DWORD IKSW2;             //Length between Shoulder and Wrist, decimals = 2
DWORD IKA14;           //Angle of the line S>W with respect to the ground in
radians, decimals = 4
DWORD IKA24;           //Angle of the line S>W with respect to the femur in
radians, decimals = 4
DWORD Temp1;
DWORD Temp2;
BYTE IKSolution;               //Output true if the solution is possible
BYTE IKSolutionWarning;        //Output true if the solution is NEARLY
possible
BYTE IKSolutionError;          //Output true if the solution is NOT possible
//-------------------------------------------------------------------
//[TIMING]
```

```
WORD wTimerWOverflowCnt;      //used in WTimer overflow. Will keep a 16 bit
overflow so we have a 32 bit timer
DWORD lCurrentTime;
DWORD lTimerStart;                  //Start time of the calculation cycles
DWORD lTimerEnd;              //End time of the calculation cycles
BYTE CycleTime;                     //Total Cycle time

WORD SSCTime;                       //Time for servo updates
WORD PrevSSCTime;             //Previous time for the servo updates

BYTE InputTimeDelay;          //Delay that depends on the input to get the
"sneaking" effect
WORD SpeedControl;                  //Adjustible Delay
//--------------------------------------------------------------------
//[GLOBAL]
BYTE HexOn;                         //Switch to turn on Phoenix
BYTE Prev_HexOn;              //Previous loop state
//--------------------------------------------------------------------
//[Balance]
BYTE BalanceMode;
SHORT TotalTransX;
SHORT TotalTransZ;
SHORT TotalTransY;
SHORT TotalYBal;
SHORT TotalXBal;
SHORT TotalZBal;
SHORT TotalY;                 //Total Y distance between the center of the
body and the feet

//[Single Leg Control]
BYTE SelectedLeg;
BYTE Prev_SelectedLeg;
SHORT SLLegX;
SHORT SLLegY;
SHORT SLLegZ;
BYTE AllDown;

//[gait]
BYTE GaitType;                      //Gait type
BYTE NomGaitSpeed;                  //Nominal speed of the gait

BYTE LegLiftHeight;                 //Current Travel height
SHORT TravelLengthX;          //Current Travel length X
SHORT TravelLengthZ;          //Current Travel length Z
SHORT TravelRotationY;        //Current Travel Rotation Y

BYTE TLDivFactor;             //Number of steps that a leg is on the floor
while walking
BYTE NrLiftedPos;             //Number of positions that a single leg is
lifted (1-3)
BYTE HalfLiftHeigth;          //If TRUE the outer positions of the ligted
legs will be half height

BYTE GaitInMotion;                  //Temp to check if the gait is in motion
BYTE StepsInGait;             //Number of steps in gait
BYTE LastLeg;                       //TRUE when the current leg is the last
leg of the sequence
```

```
BYTE GaitStep;                      //Actual Gait step

BYTE GaitLegNr[6];                  //Init position of the leg

BYTE GaitLegNrIn;             //Input Number of the leg

CHAR GaitPosX[6];             //Array containing Relative X position
corresponding to the Gait
CHAR GaitPosY[6];             //Array containing Relative Y position
corresponding to the Gait
CHAR GaitPosZ[6];             //Array containing Relative Z position
corresponding to the Gait
CHAR GaitRotY[6];             //Array containing Relative Y rotation
corresponding to the Gait

int tempVariable = 0; // TEMPORARY
volatile uint16_t tempSR = 0;
volatile uint16_t tempCNT = 0;
volatile uint16_t tempCR = 0;

//// TO DO!  (down)
////====================================================================////=
====================================================================
////[INIT]
//
////Checks SSC version number if it ends with "GP"
////enable the GP player if it does
//delay_ms(10);
//GPEnable=0
//serout cSSC_OUT, cSSC_BAUD, ["ver", 13]
//
//GetSSCVersion:
//serin cSSC_IN, cSSC_BAUD, 1000, timeout, [GPVerData(Index)]
//Index = (Index+1)//3 // shift last 3 chars in data
//goto GetSSCVersion
//
//timeout:
//if (GPVerData(0) + GPVerData(1) + GPVerData(2)) = 164 then // Check if the
last 3 chars are G(71) P(80) cr(13)
//  GPEnable = 1
//else
//  sound P9, [40\5000,40\5000]
//endif
//delay_ms(10);
//
//// TO DO (up)

void InitHexCrawler(void)
{
      // Initialize necessary system oomponents
      delay_init(72);
      usart2_init();
      tim3_Init();

      /* Deprecated
      // Turn off all LEDs
      LedA = 0;
      LedB = 0;
      LedC = 0;
      Eyes = 0;
      */

      // Tars Init Positions
      for (LegIndex  = 0 ; LegIndex < 6 ; LegIndex++) // Set start positions
for each leg
      {
            LegPosX[LegIndex] = cInitPosX[LegIndex];
            LegPosY[LegIndex] = cInitPosY[LegIndex];
            LegPosZ[LegIndex] = cInitPosZ[LegIndex];
      }

      // Single leg control. Make sure no leg is selected
      SelectedLeg = 255; // No Leg selected
      Prev_SelectedLeg = 255;

      // Body Positions
      BodyPosX = 0;
      BodyPosY = 0;
      BodyPosZ = 0;

      // Body Rotations
      BodyRotX = 0;
      BodyRotY = 0;
      BodyRotZ = 0;

      // Gait
      GaitType = 0;
      BalanceMode = 0;
      LegLiftHeight = 50;
      GaitStep = 1;

      GaitSelect();

      //Timer
      #define WTIMERTICSPERMS  2000// we have 16 clocks per ms and we are
incrementing every 8 so divide again by 2
      lCurrentTime = 0 ;

      //Initialize Controller
      //TO DO
      //InitController();

      //Initialize Servo Packet
      pac.packet.SC = 0x215343;       //!SC
      pac.packet.B1 = 0;                    //Ramp Speed = 0
      pac.packet.CR = 0x0D;           //CR

      //SSC
      SSCTime = 150;
      HexOn = 1;

}

/******************************
```

```
 * Main application entry point.
 */
int main (void)
{
      // Initialize system compoments
      SystemInit();
      InitHexCrawler();
      InitializeLED();        // Initialize the LEDs

      // TESTING
      tempVariable = GetCurrentTime();
      while (1)
      {
            delay_ms(1000);
            tempVariable = GetCurrentTime();
            SetAllLEDOn();

            delay_ms(1000);
            tempVariable = GetCurrentTime();
            SetAllLEDOff();
      }
      // END TESTING


      // Loop forever
      while (1)
      {
            // Start time
            lTimerStart = GetCurrentTime();

            // Read input from controller
            // TODO
            //ControlInput();

            // Read the current state of the buttons
            // ReadButtons();  // Deprecated - there are no buttons currently
configured on STM32F107 board

            // WriteOutputs();      // Deprecated

        //GP Player
        if (GPEnable) {
            //GPPlayer();
        }

        //Single leg control
        SingleLegControl();

        //Gait
        GaitSeq();

        //Balance calculations
        TotalTransX = 0; //reset values used for calculation of balance
        TotalTransZ = 0;
        TotalTransY = 0;
        TotalXBal = 0;
        TotalYBal = 0;
        TotalZBal = 0;

        if (BalanceMode > 0) {
            // balance calculations for all Right legs
            for (LegIndex = 0 ; LegIndex < 3 ; LegIndex++) {
                  BalCalcOneLeg(-LegPosX[LegIndex] + GaitPosX[LegIndex],
                                                LegPosZ[LegIndex] +
GaitPosZ[LegIndex],
                                                (LegPosY[LegIndex] -
cInitPosY[LegIndex]) + GaitPosY[LegIndex],
                                                LegIndex);
          }
          // balance calculations for all Left legs
          for (LegIndex = 3 ; LegIndex < 6 ; LegIndex++) {
            BalCalcOneLeg(LegPosX[LegIndex] + GaitPosX[LegIndex],
                                      LegPosZ[LegIndex] + GaitPosZ[LegIndex],
                                      (LegPosY[LegIndex] -
cInitPosY[LegIndex]) + GaitPosY[LegIndex],
                                      LegIndex);
          }
            BalanceBody();
        }

        //Reset IKsolution indicators
        IKSolution = 0;
        IKSolutionWarning = 0;
        IKSolutionError = 0;

        //Do IK for all Right legs
        for (LegIndex = 0 ; LegIndex < 3 ; LegIndex++) {
              BodyIK(-LegPosX[LegIndex] + BodyPosX + GaitPosX[LegIndex] -
TotalTransX,
                          LegPosZ[LegIndex] + BodyPosZ + GaitPosZ[LegIndex] -
TotalTransZ,
                          LegPosY[LegIndex] + BodyPosY+GaitPosY[LegIndex] -
TotalTransY, GaitRotY[LegIndex], LegIndex);
              LegIK(LegPosX[LegIndex] - BodyPosX + BodyIKPosX -
(GaitPosX[LegIndex] - TotalTransX),
                          LegPosY[LegIndex] + BodyPosY - BodyIKPosY +
GaitPosY[LegIndex] - TotalTransY,
                          LegPosZ[LegIndex] + BodyPosZ - BodyIKPosZ +
GaitPosZ[LegIndex] - TotalTransZ, LegIndex);
        }

        //Do IK for all Left legs
        for (LegIndex = 3 ; LegIndex < 6 ; LegIndex++) {
              BodyIK(LegPosX[LegIndex] - BodyPosX + GaitPosX[LegIndex] -
TotalTransX,
                           LegPosZ[LegIndex] + BodyPosZ+GaitPosZ[LegIndex] -
TotalTransZ,
                           LegPosY[LegIndex] + BodyPosY+GaitPosY[LegIndex] -
TotalTransY,
                           GaitRotY[LegIndex], LegIndex);
              LegIK(LegPosX[LegIndex] + BodyPosX - BodyIKPosX +
GaitPosX[LegIndex] - TotalTransX,
                         LegPosY[LegIndex] + BodyPosY - BodyIKPosY +
GaitPosY[LegIndex] - TotalTransY,
                         LegPosZ[LegIndex] + BodyPosZ - BodyIKPosZ +
GaitPosZ[LegIndex] - TotalTransZ, LegIndex);
        }

        //Check mechanical limits
        CheckAngles();

        //Write IK errors to leds
        LedC = IKSolutionWarning;
        LedA = IKSolutionError;

        //Drive Servos
        if (HexOn) {
          if (HexOn && (Prev_HexOn == 0)) {
// TO DO Sound?
//          Sound(P9,[60\4000,80\4500,100\5000]);
            Eyes = 1;
            }

          //Set SSC time
            if ((fabs(TravelLengthX) > cTravelDeadZone) |
(fabs(TravelLengthZ) > cTravelDeadZone) | (fabs(TravelRotationY * 2) >
cTravelDeadZone)) {
              SSCTime = NomGaitSpeed + (InputTimeDelay*2) + SpeedControl;

              //Add aditional delay when Balance mode is on
            if ( BalanceMode ) {
                SSCTime = SSCTime + 100;
            }

            else //Movement speed excl. Walking
              SSCTime = 200 + SpeedControl;
            }

          //Sync BAP with SSC while walking to ensure the prev is completed
before sending the next one
            if (GaitPosX[cRF] || GaitPosX[cRM] || GaitPosX[cRR] ||
GaitPosX[cLF] || GaitPosX[cLM] || GaitPosX[cLR]  ||
                  GaitPosY[cRF] || GaitPosY[cRM] || GaitPosY[cRR] ||
GaitPosY[cLF] || GaitPosY[cLM] || GaitPosY[cLR]  ||
                  GaitPosZ[cRF] || GaitPosZ[cRM] || GaitPosZ[cRR] ||
GaitPosZ[cLF] || GaitPosZ[cLM] || GaitPosZ[cLR]  ||
                  GaitRotY[cRF] || GaitRotY[cRM] || GaitRotY[cRR] ||
GaitRotY[cLF] || GaitRotY[cLM] || GaitRotY[cLR]) {

              //Get endtime and calculate wait time
            lTimerEnd = GetCurrentTime();
            CycleTime = (lTimerEnd - lTimerStart) / WTIMERTICSPERMS;

              //Wait for previous commands to be completed while walking
              delay_ms(min((PrevSSCTime - CycleTime - 45), 1)); //      Min 1
ensures that there always is a value in the delay_ms command
            }

            ServoDriver();

        } else {
          //Turn the bot off
          if (Prev_HexOn || !AllDown ) {
            SSCTime = 600;
            ServoDriver();
              //TO DO sound?
//          Sound P9,[100\5000,80\4500,60\4000];
            delay_ms(600);
          } else {
              FreeServos();
              Eyes = 0;
          }
        }

        //Store previous HexOn State
        if (HexOn) {
          Prev_HexOn = 1;
        } else {
          Prev_HexOn = 0;
        }
      }
}
//dead:
//goto dead
//======================================================================
//[ReadButtons] Reading input buttons from the ABB
void ReadButtons(void) {

      prev_butA = butA;
      prev_butB = butB;
      prev_butC = butC;

//    if (S1Pressed()) {
//          HexOn = 1;
//    }
//    if (S2Pressed()) {
//          HexOn = 0;
//    }

      butA = S1Pressed();
      butB = S2Pressed();
      butC = S3Pressed();
}
//----------------------------------------------------------------------
//[WriteOutputs] Updates the state of the leds
void WriteOutputs(void) {
      if (LedA) {
            GPIOD->ODR &= 0x0004;   // LED A on Pin D02
      } else {
            GPIOD->ODR |= 0xFFFB;
      }

      if (LedB) {
            GPIOD->ODR &= 0x0008;   // LED B on Pin D03
      } else {
            GPIOD->ODR |= 0xFFF7;
      }
      if (LedC) {
            GPIOD->ODR &= 0x0010;   // LED C on Pin D04
      } else {
            GPIOD->ODR |= 0xFFEF;
      }

      if (Eyes) {
      GPIOD->ODR &= 0x0080;   // LED D on Pin D07
      } else {
      GPIOD->ODR |= 0xFF7F;
      }
}
//----------------------------------------------------------------------
//[GP PLAYER]
//TO DO
//GPStatSeq             var byte
//GPStatFromStep  var byte
//GPStatToStep    var byte
//GPStatTime            var byte
//GPPlayer:
//
//  //Start sequence
//  if (GPStart=1) {
//    serout cSSC_OUT, cSSC_BAUD, ["PL0SQ", dec GPSeq, "ONCE", 13] //Start
sequence
//
//    //Wait for GPPlayer to complete sequence
//GPWait:
//    serout cSSC_OUT, cSSC_BAUD, ["QPL0", 13]
//    serin cSSC_IN, cSSC_BAUD, [GPStatSeq, GPStatFromStep, GPStatToStep,
GPStatTime]
//
//    if (GPStatSeq<>255 | GPStatFromStep<>0 | GPStatToStep<>0 |
GPStatTime<>0) {
//      GOTO GPWait //Continue waiting
//    ENDIF
//
//    GPStart=0
//  ENDIF
//
//return
//----------------------------------------------------------------------
//[SINGLE LEG CONTROL]
void SingleLegControl (void) {

  //Check if all legs are down
  AllDown = ((LegPosY[cRF] == cInitPosY[cRF]) & (LegPosY[cRM] ==
cInitPosY[cRM]) & (LegPosY[cRR] == cInitPosY[cRR]) & (LegPosY[cLR] ==
cInitPosY[cLR]) & (LegPosY[cLM] == cInitPosY[cLM]) & (LegPosY[cLF] ==
cInitPosY[cLF]));
  // pointless comparison of unsigned integer removed:
"( SelectedLeg >= 0 )"
  if (SelectedLeg <= 5) {
    if (SelectedLeg != Prev_SelectedLeg) {

      if (AllDown) { //Lift leg a bit when it got selected
```

```
        LegPosY[SelectedLeg] = cInitPosY[SelectedLeg] - 20;

            //Store current status
            Prev_SelectedLeg = SelectedLeg;

      } else { //Return prev leg back to the init position
          LegPosX[Prev_SelectedLeg] = cInitPosX[Prev_SelectedLeg];
          LegPosY[Prev_SelectedLeg] = cInitPosY[Prev_SelectedLeg];
          LegPosZ[Prev_SelectedLeg] = cInitPosZ[Prev_SelectedLeg];
      }

    } else if (!SLHold) {
      LegPosY[SelectedLeg] = LegPosY[SelectedLeg] + SLLegY;
      LegPosX[SelectedLeg] = cInitPosX[SelectedLeg] + SLLegX;
      LegPosZ[SelectedLeg] = cInitPosZ[SelectedLeg] + SLLegZ;
    }


  } else { //All legs to init position
    if ( !AllDown ) {
      for ( LegIndex = 0 ; LegIndex < 6 ; LegIndex++ ) {
          LegPosX[LegIndex] = cInitPosX[LegIndex];
          LegPosY[LegIndex] = cInitPosY[LegIndex];
          LegPosZ[LegIndex] = cInitPosZ[LegIndex];
      }
    }
    if ( Prev_SelectedLeg != 255 ) {
      Prev_SelectedLeg = 255;
    }
  }
}
//---------------------------------------------------------------------
void GaitSelect (void) {
  //Gait selector
  if (GaitType == 0) { //Ripple Gait 6 steps
      GaitLegNr[cLR] = 1;
      GaitLegNr[cRF] = 2;
      GaitLegNr[cLM] = 3;
      GaitLegNr[cRR] = 4;
      GaitLegNr[cLF] = 5;
      GaitLegNr[cRM] = 6;

      NrLiftedPos = 1;
      HalfLiftHeigth = 0;
      TLDivFactor = 4;
      StepsInGait = 6;
      NomGaitSpeed = 100;
  }

  if (GaitType == 1) { //Ripple Gait 12 steps
      GaitLegNr[cLR] = 1;
      GaitLegNr[cRF] = 3;
      GaitLegNr[cLM] = 5;
      GaitLegNr[cRR] = 7;
      GaitLegNr[cLF] = 9;
      GaitLegNr[cRM] = 11;
```

```
        NrLiftedPos = 3;
        HalfLiftHeigth = 0;//1
        TLDivFactor = 8;
        StepsInGait = 12;
      NomGaitSpeed = 85;
    }

    if (GaitType == 2) { //Quadripple 9 steps
        GaitLegNr[cLR] = 1;
        GaitLegNr[cRF] = 2;
        GaitLegNr[cLM] = 4;
        GaitLegNr[cRR] = 5;
        GaitLegNr[cLF] = 7;
        GaitLegNr[cRM] = 8;

        NrLiftedPos = 2;
        HalfLiftHeigth = 0;
        TLDivFactor = 6;
        StepsInGait = 9;
      NomGaitSpeed = 100;
    }

    if (GaitType == 3) { //Tripod 4 steps
        GaitLegNr[cLR] = 3;
        GaitLegNr[cRF] = 1;
        GaitLegNr[cLM] = 1;
        GaitLegNr[cRR] = 1;
        GaitLegNr[cLF] = 3;
        GaitLegNr[cRM] = 3;

        NrLiftedPos = 1;
        HalfLiftHeigth = 0;
        TLDivFactor = 2;
        StepsInGait = 4;
      NomGaitSpeed = 150;
    }

    if (GaitType == 4) { //Tripod 6 steps
        GaitLegNr[cLR] = 4;
        GaitLegNr[cRF] = 1;
        GaitLegNr[cLM] = 1;
        GaitLegNr[cRR] = 1;
        GaitLegNr[cLF] = 4;
        GaitLegNr[cRM] = 4;

        NrLiftedPos = 2;
        HalfLiftHeigth = 0;
        TLDivFactor = 4;
        StepsInGait = 6;
      NomGaitSpeed = 100;
    }

    if (GaitType == 5) { //Tripod 8 steps
        GaitLegNr[cLR] = 5;
        GaitLegNr[cRF] = 1;
        GaitLegNr[cLM] = 1;
        GaitLegNr[cRR] = 1;
        GaitLegNr[cLF] = 5;
        GaitLegNr[cRM] = 5;

        NrLiftedPos = 3;
        HalfLiftHeigth = 1;
        TLDivFactor = 4;
        StepsInGait = 8;
      NomGaitSpeed = 85;
    }

    if (GaitType == 6) { //Wave 12 steps
        GaitLegNr[cLR] = 1;
        GaitLegNr[cRF] = 11;
        GaitLegNr[cLM] = 3;
        GaitLegNr[cRR] = 7;
        GaitLegNr[cLF] = 5;
        GaitLegNr[cRM] = 9;

        NrLiftedPos = 1;
        HalfLiftHeigth = 0;
        TLDivFactor = 10;
        StepsInGait = 12;
      NomGaitSpeed = 85;
    }

    if (GaitType == 7) { //Wave 18 steps
        GaitLegNr[cLR] = 4;
        GaitLegNr[cRF] = 1;
        GaitLegNr[cLM] = 7;
        GaitLegNr[cRR] = 13;
        GaitLegNr[cLF] = 10;
        GaitLegNr[cRM] = 16;

        NrLiftedPos = 2;
        HalfLiftHeigth = 0;
        TLDivFactor = 16;
        StepsInGait = 18;
      NomGaitSpeed = 85;
    }
  }
  //--------------------------------------------------------------------
  //[GAIT Sequence]
  void GaitSeq (void) {
    //Calculate Gait sequence
    LastLeg = 0;
    for (LegIndex = 0 ; LegIndex < 6 ; LegIndex++) { // for all legs

      if (LegIndex == 5) { // last leg
        LastLeg = 1;
      }

      Gait(LegIndex);
    }   // next leg
  }
  //--------------------------------------------------------------------
  //[GAIT]
  //GaitCurrentLegNr var nib
```

```
void Gait(BYTE GaitCurrentLegNr) {

  //Check IF the Gait is in motion
  GaitInMotion = ((fabs(TravelLengthX) > cTravelDeadZone) |
(fabs(TravelLengthZ) > cTravelDeadZone) | (fabs(TravelRotationY) >
cTravelDeadZone));

  //Clear values under the cTravelDeadZone
  if (GaitInMotion == 0) {
    TravelLengthX = 0;
    TravelLengthZ = 0;
    TravelRotationY = 0;
  }

  //Leg middle up position
       //Gait in motion
                         Gait NOT in motion, return to home position
  if ((GaitInMotion & (NrLiftedPos == 1 | NrLiftedPos == 3) & GaitStep ==
GaitLegNr[GaitCurrentLegNr]) | (!GaitInMotion & GaitStep ==
GaitLegNr[GaitCurrentLegNr] & ((fabs(GaitPosX[GaitCurrentLegNr]) > 2) |
(fabs(GaitPosZ[GaitCurrentLegNr]) > 2) | (fabs(GaitRotY[GaitCurrentLegNr]) >
2)))) {     //Up
    GaitPosX[GaitCurrentLegNr] = 0;
    GaitPosY[GaitCurrentLegNr] = -LegLiftHeight;
    GaitPosZ[GaitCurrentLegNr] = 0;
    GaitRotY[GaitCurrentLegNr] = 0;
  } else {

    //Optional Half heigth Rear
    if (((NrLiftedPos == 2 & GaitStep == GaitLegNr[GaitCurrentLegNr]) |
(NrLiftedPos == 3 & (GaitStep == GaitLegNr[GaitCurrentLegNr] - 1 | GaitStep
== GaitLegNr[GaitCurrentLegNr] + (StepsInGait - 1)))) & GaitInMotion) {
        GaitPosX[GaitCurrentLegNr] = -TravelLengthX / 2;
      GaitPosY[GaitCurrentLegNr] = -LegLiftHeight / (HalfLiftHeigth + 1);
      GaitPosZ[GaitCurrentLegNr] = -TravelLengthZ / 2;
      GaitRotY[GaitCurrentLegNr] = -TravelRotationY / 2;
      } else {

        //Optional half heigth front
      if ((NrLiftedPos >= 2) & (GaitStep == GaitLegNr[GaitCurrentLegNr] + 1 |
GaitStep == GaitLegNr[GaitCurrentLegNr] - (StepsInGait - 1)) & GaitInMotion)
{
        GaitPosX[GaitCurrentLegNr] = TravelLengthX / 2;
        GaitPosY[GaitCurrentLegNr] = -LegLiftHeight / (HalfLiftHeigth+1);
        GaitPosZ[GaitCurrentLegNr] = TravelLengthZ / 2;
        GaitRotY[GaitCurrentLegNr] = TravelRotationY / 2;
      } else {

            //Leg front down position
            if ((GaitStep == GaitLegNr[GaitCurrentLegNr] + NrLiftedPos |
GaitStep == GaitLegNr[GaitCurrentLegNr] - (StepsInGait - NrLiftedPos)) &
GaitPosY[GaitCurrentLegNr] < 0) {
            GaitPosX[GaitCurrentLegNr] = TravelLengthX / 2;
          GaitPosZ[GaitCurrentLegNr] = TravelLengthZ / 2;
          GaitRotY[GaitCurrentLegNr] = TravelRotationY / 2;
          GaitPosY[GaitCurrentLegNr] = 0; //Only move leg down at once if
terrain adaption is turned off
```

```
            //Move body forward
            } else {
          GaitPosX[GaitCurrentLegNr] = GaitPosX[GaitCurrentLegNr] -
(TravelLengthX / TLDivFactor);
          GaitPosY[GaitCurrentLegNr] = 0;
          GaitPosZ[GaitCurrentLegNr] = GaitPosZ[GaitCurrentLegNr] -
(TravelLengthZ / TLDivFactor);
          GaitRotY[GaitCurrentLegNr] = GaitRotY[GaitCurrentLegNr] -
(TravelRotationY / TLDivFactor);
        }
      }
    }
  }

  //Advance to the next step
  if (LastLeg) {  //The last leg in this step
    GaitStep = GaitStep + 1;
    if (GaitStep > StepsInGait) {
      GaitStep = 1;
    }
  }
}
//--------------------------------------------------------------------
//[BalCalcOneLeg]
//BalLegNr var nib
void BalCalcOneLeg(SHORT PosX, SHORT PosZ, SHORT PosY, BYTE BalLegNr) {
  //Calculating totals from center of the body to the feet
  TotalZ = cOffsetZ[BalLegNr] + PosZ;
  TotalX = cOffsetX[BalLegNr] + PosX;
  TotalY = 150 + PosY; // using the value 150 to lower the centerpoint of
rotation 'BodyPosY +
  TotalTransY = TotalTransY + PosY;
  TotalTransZ = TotalTransZ + TotalZ;
  TotalTransX = TotalTransX + TotalX;

  GetAtan2(TotalX, TotalZ);
  TotalYBal =  TotalYBal + (Atan4 * 180) / 31415;

  GetAtan2(TotalX, TotalY);
  TotalZBal = TotalZBal + ((Atan4 * 180) / 31415) -90; //Rotate balance
circle 90 deg

  GetAtan2(TotalZ, TotalY);
  TotalXBal = TotalXBal + ((Atan4 * 180) / 31415) - 90; //Rotate balance
circle 90 deg

}
//--------------------------------------------------------------------
//[BalanceBody]
void BalanceBody(void) {
      TotalTransZ = TotalTransZ / 6;
      TotalTransX = TotalTransX / 6;
      TotalTransY = TotalTransY / 6;

      if (TotalYBal > 0) {           //Rotate balance circle by +/- 180 deg
            TotalYBal = TotalYBal - 180;
      } else {
            TotalYBal = TotalYBal + 180;
      }
      if (TotalZBal < -180) { //Compensate for extreme balance positions that
causes owerflow
            TotalZBal = TotalZBal + 360;
      }

      if (TotalXBal < -180) {   //Compensate for extreme balance positions
that causes owerflow
            TotalXBal = TotalXBal + 360;
      }

      //Balance rotation
      TotalYBal = -TotalYBal / 6;
      TotalXBal = -TotalXBal / 6;
      TotalZBal = TotalZBal / 6;

}
//--------------------------------------------------------------------
//[GETSINCOS] Get the sinus and cosinus from the angle +/- multiple circles
//AngleDeg1       - Input Angle in degrees
//Sin4      - Output Sinus of AngleDeg
//Cos4            - Output Cosinus of AngleDeg
void GetSinCos(SHORT AngleDeg1) {
      //Get the absolute value of AngleDeg
      if (AngleDeg1 < 0) {
        ABSAngleDeg1 = AngleDeg1 * -1;
      } else {
        ABSAngleDeg1 = AngleDeg1;
      }

      //Shift rotation to a full circle of 360 deg -> AngleDeg // 360
      if (AngleDeg1 < 0) {    //Negative values
            AngleDeg1 = 3600 - (ABSAngleDeg1 - (3600 * (ABSAngleDeg1 /
3600)));
      } else {                      //Positive values
            AngleDeg1 = ABSAngleDeg1 - (3600 * (ABSAngleDeg1 / 3600));
      }

      if (AngleDeg1 >= 0 && AngleDeg1 <= 900) { // 0 to 90 deg
            sin4 = GetSin[AngleDeg1 / 5];                   // 5 is the
presision (0.5) of the table
            cos4 = GetSin[(900 - (AngleDeg1)) / 5];

      } else if ((AngleDeg1 > 900) && (AngleDeg1 <= 1800)) {      // 90 to
180 deg
            sin4 = GetSin[(900 - (AngleDeg1 - 900)) / 5]; // 5 is the
presision (0.5) of the table
            cos4 = -GetSin[(AngleDeg1 - 900) / 5];

      } else if ((AngleDeg1 > 1800) && (AngleDeg1 <= 2700)) { // 180 to 270
deg
            sin4 = -GetSin[(AngleDeg1 - 1800) / 5];   // 5 is the presision
(0.5) of the table
            cos4 = -GetSin[(2700 - AngleDeg1) / 5];
      } else if ((AngleDeg1 > 2700) && (AngleDeg1 <= 3600)) { // 270 to 360
deg
            sin4 = -GetSin[(3600 - AngleDeg1) / 5]; // 5 is the presision
(0.5) of the table
            cos4 = GetSin[(AngleDeg1 - 2700) / 5];
      }
}
//----------------------------------------------------------------------
//[GETARCCOS] Get the sinus and cosinus from the angle +/- multiple circles
//Cos4      - Input Cosinus
//AngleRad4       - Output Angle in AngleRad4
SHORT GetArcCos(SHORT Cos4) {
  //Check for negative value
  if (Cos4 < 0) {
    Cos4 = -Cos4;
    NegativeValue = 1;
  } else {
    NegativeValue = 0;
  }

  //Limit Cos4 to its maximal value
  Cos4 = max(Cos4, c4DEC);

  if ((Cos4 >= 0) && (Cos4 < 9000)) {
    AngleRad4 = GetACos[Cos4 / 79]; //79=table resolution (1/127)
    AngleRad4 = AngleRad4 * 616 / c1DEC; //616=acos resolution (pi/2/255)

  } else if ((Cos4 >= 9000) && (Cos4 < 9900)) {
    AngleRad4 = GetACos[((Cos4 - 9000) / 8) + 114]; //8=table resolution
(0.1/127), 114 start address 2nd bytetable range
    AngleRad4 = AngleRad4 * 616 / c1DEC; //616=acos resolution (pi/2/255)

  } else if ((Cos4 >= 9900) && (Cos4 <= 10000)) {
    AngleRad4 = GetACos[((Cos4 - 9900) / 2) + 227]; //2=table resolution
(0.01/64), 227 start address 3rd bytetable range
    AngleRad4 = AngleRad4 * 616 / c1DEC; //616=acos resolution (pi/2/255)
  }

  //Add negative sign
  if (NegativeValue) {
    AngleRad4 = 31416 - AngleRad4;
  }

  return AngleRad4;
}
//----------------------------------------------------------------------
//[GETATAN2] Simplyfied ArcTan2 function based on fixed point ArcCos
//ArcTanX         - Input X
//ArcTanY         - Input Y
//ArcTan4         - Output ARCTAN2(X/Y)
//XYhyp2                - Output presenting Hypotenuse of X and Y
SHORT GetAtan2(SHORT AtanX, SHORT AtanY) {
  XYhyp2 = sqrt((AtanX * AtanX * c4DEC) + (AtanY * AtanY * c4DEC));
  GetArcCos(AtanX * c6DEC / XYhyp2);

  Atan4 = AngleRad4 * (AtanY / fabs(AtanY)); //Add sign
  return Atan4;
}
//--------------------------------------------------------------------
//[BODY INVERSE KINEMATICS]
//BodyRotX         - Global Input pitch of the body
//BodyRotY         - Global Input rotation of the body
//BodyRotZ         - Global Input roll of the body
//RotationY         - Input Rotation for the gait
//PosX            - Input position of the feet X
//PosZ            - Input position of the feet Z
//SinB                  - Sin buffer for BodyRotX
//CosB            - Cos buffer for BodyRotX
//SinG                  - Sin buffer for BodyRotZ
//CosG            - Cos buffer for BodyRotZ
//BodyIKPosX          - Output Position X of feet with Rotation
//BodyIKPosY          - Output Position Y of feet with Rotation
//BodyIKPosZ          - Output Position Z of feet with Rotation

void BodyIK(SHORT PosX, SHORT PosZ, SHORT PosY, CHAR RotationY, BYTE
BodyIKLeg) {

  //Calculating totals from center of the body to the feet
  TotalZ = cOffsetZ[BodyIKLeg] + PosZ;
  TotalX = cOffsetX[BodyIKLeg] + PosX;
  //PosY are equal to a "TotalY"

  //Successive global rotation matrix:
  //Math shorts for rotation: Alfa (A) = Xrotate, Beta (B) = Zrotate, Gamma
(G) = Yrotate
  //Sinus Alfa = sinA, cosinus Alfa = cosA. and so on...

  //First calculate sinus and cosinus for each rotation:
  GetSinCos((BodyRotX + TotalXBal) * c1DEC);
  sinG4 = sin4;
  cosG4 = cos4;

  GetSinCos((BodyRotZ + TotalZBal) * c1DEC);
  sinB4 = sin4;
  cosB4 = cos4;

  GetSinCos((BodyRotY + RotationY + TotalYBal) * c1DEC);
  sinA4 = sin4;
  cosA4 = cos4;

  //Calcualtion of rotation matrix:
  //BodyIKPosX = TotalX - (TotalX*CosA*CosB - TotalZ*CosB*SinA + PosY*SinB)
  //BodyIKPosZ = TotalZ - (TotalX*CosG*SinA + TotalX*CosA*SinB*SinG +
TotalZ*CosA*CosG - TotalZ*SinA*SinB*SinG - PosY*CosB*SinG)
  //BodyIKPosY = PosY   - (TotalX*SinA*SinG - TotalX*CosA*CosG*SinB +
TotalZ*CosA*SinG + TotalZ*CosG*SinA*SinB + PosY*CosB*CosG)
  BodyIKPosX = ((TotalX * c2DEC) - ((TotalX * c2DEC * cosA4 / c4DEC * cosB4 /
c4DEC) - (TotalZ * c2DEC * cosB4 / c4DEC * sinA4 / c4DEC) + PosY * c2DEC *
sinB4 / c4DEC )) / c2DEC;
  BodyIKPosZ = ((TotalZ * c2DEC) - ((TotalX * c2DEC * cosG4 / c4DEC * sinA4 /
c4DEC) + (TotalX * c2DEC * cosA4 / c4DEC * sinB4 / c4DEC * sinG4 / c4DEC) +
(TotalZ * c2DEC * cosA4 / c4DEC * cosG4 / c4DEC) - (TotalZ * c2DEC * sinA4 /
c4DEC * sinB4 / c4DEC * sinG4 / c4DEC) - (PosY * c2DEC * cosB4 / c4DEC *
sinG4 / c4DEC))) / c2DEC;
  BodyIKPosY = ((PosY * c2DEC) - ((TotalX * c2DEC * sinA4 / c4DEC * sinG4 /
c4DEC) - (TotalX * c2DEC * cosA4 / c4DEC * cosG4 / c4DEC * sinB4 / c4DEC) +
(TotalZ * c2DEC * cosA4 / c4DEC * sinG4 / c4DEC) + (TotalZ * c2DEC * cosG4 /
c4DEC * sinA4 / c4DEC * sinB4 / c4DEC) + PosY * c2DEC * cosB4 / c4DEC * cosG4
/ c4DEC)) / c2DEC;

}
//---------------------------------------------------------------------
//[LEG INVERSE KINEMATICS] Calculates the angles of the coxa, femur and tibia
for the given position of the feet
//IKFeetPosX                 - Input position of the Feet X
//IKFeetPosY                 - Input position of the Feet Y
//IKFeetPosZ                 - Input Position of the Feet Z
//IKSolution                 - Output true IF the solution is possible
//IKSolutionWarning     - Output true IF the solution is NEARLY possible
//IKSolutionError - Output true IF the solution is NOT possible
//FemurAngle1           - Output Angle of Femur in degrees
//TibiaAngle1           - Output Angle of Tibia in degrees
//CoxaAngle1                 - Output Angle of Coxa in degrees
//LegIKLegNr var nib
void LegIK (SHORT IKFeetPosX, SHORT IKFeetPosY, SHORT IKFeetPosZ, BYTE
LegIKLegNr) {

      //Calculate IKCoxaAngle and IKFeetPosXZ
      GetAtan2(IKFeetPosX, IKFeetPosZ);
      CoxaAngle1[LegIKLegNr] = ((Atan4 * 180) / 3141) +
CoxaAngle1[LegIKLegNr];

      //Length between the Coxa and tars (foot)
      IKFeetPosXZ = XYhyp2 / c2DEC;

      //Using GetAtan2 for solving IKA1 and IKSW
      //IKA14 - Angle between SW line and the ground in radians
      IKA14 = GetAtan2(IKFeetPosY, IKFeetPosXZ - cCoxaLength);
      //IKSW2 - Length between femur axis and tars
      IKSW2 = XYhyp2;

      //IKA2 - Angle of the line S>W with respect to the femur in radians
     Temp1 = (((cFemurLength * cFemurLength) - (cTibiaLength * cTibiaLength))
* c4DEC + (IKSW2 * IKSW2));
      Temp2 = ((2 * cFemurLength) * c2DEC * IKSW2);
      IKA24 = GetArcCos(Temp1 / (Temp2 / c4DEC));

      //IKFemurAngle
      FemurAngle1[LegIKLegNr] = -(IKA14 + IKA24) * 180 / 3141 + 900;

      //IKTibiaAngle
     Temp1 = (((cFemurLength * cFemurLength) + (cTibiaLength * cTibiaLength))
* c4DEC - (IKSW2 * IKSW2));
      Temp2 = (2 * cFemurLength * cTibiaLength);
      GetArcCos(Temp1 / Temp2);
      TibiaAngle1[LegIKLegNr] = -(900 - AngleRad4 * 180 / 3141);

      //Set the Solution quality
      if(IKSW2 < (cFemurLength + cTibiaLength - 30) * c2DEC) {
            IKSolution = 1;
      } else {
            if(IKSW2 < (cFemurLength + cTibiaLength) * c2DEC) {
                  IKSolutionWarning = 1;
            } else {
                  IKSolutionError = 1;
            }
      }
}
//----------------------------------------------------------------------
//[CHECK ANGLES] Checks the mechanical limits of the servos
void CheckAngles(void) {

  for (LegIndex = 0 ; LegIndex < 6 ; LegIndex++) {
    CoxaAngle1[LegIndex]  = max(min(CoxaAngle1[LegIndex],
cCoxaMin1[LegIndex]), cCoxaMax1[LegIndex]);
    FemurAngle1[LegIndex] = max(min(FemurAngle1[LegIndex],
cFemurMin1[LegIndex]), cFemurMax1[LegIndex]);
    TibiaAngle1[LegIndex] = max(min(TibiaAngle1[LegIndex],
cTibiaMin1[LegIndex]), cTibiaMax1[LegIndex]);
  }
}
//----------------------------------------------------------------------
//[SERVO DRIVER] Updates the positions of the servos
void ServoDriver(void) {

      //Update Right Legs
      for (LegIndex = 0 ; LegIndex < 3 ; LegIndex++) {
      SendPacket(cCoxaPin[LegIndex], cCoxaMiddle[LegIndex]); //(-
CoxaAngle1[LegIndex] + 900) * 1000 / 1059 + 650);
            SendPacket(cFemurPin[LegIndex], cFemurMiddle[LegIndex]); //(-
FemurAngle1[LegIndex] + 900) * 1000 / 1059 + 650);
      SendPacket(cTibiaPin[LegIndex], cTibiaMiddle[LegIndex]); //(-
TibiaAngle1[LegIndex] + 900) * 1000 / 1059 + 650);
      }

      //Update Left Legs
      for (LegIndex = 3 ; LegIndex < 6 ; LegIndex++) {
      SendPacket(cCoxaPin[LegIndex], cCoxaMiddle[LegIndex]);
//(CoxaAngle1[LegIndex] + 900) * 1000 / 1059 + 650);
      SendPacket(cFemurPin[LegIndex], cFemurMiddle[LegIndex]);
//(FemurAngle1[LegIndex] + 900) * 1000 / 1059 + 650);
      SendPacket(cTibiaPin[LegIndex], cTibiaMiddle[LegIndex]);
//(TibiaAngle1[LegIndex] + 900) * 1000 / 1059 + 650);
      }

      PrevSSCTime = SSCTime;
}
//----------------------------------------------------------------------
//[SEND PACKET] Sends Serial Packet to PSC to command servo position
void SendPacket(BYTE Channel, WORD Position) {
      int i = 0;

      pac.packet.B0 = Channel;
      pac.packet.B1 = 0;                    //Ramp Speed = 0
      pac.packet.B2 = LOWBYTE(Position);
      pac.packet.B3 = HIGHBYTE(Position);
      for(i = 7 ; i >= 0 ; i--) {

            ser_putbyte(pac.stream[i]);
      }

      delay_ms(23);
}
//----------------------------------------------------------------------
//[FREE SERVOS] Frees all the servos
void FreeServos(void) {
      int i = 0;

      pac.packet.B0 = 'P';
      pac.packet.B1 = 'S';
      pac.packet.B2 = 'D';

      for (LegIndex = 0 ; LegIndex < 32; LegIndex++) {

            pac.packet.B3 = LegIndex;

            for(i = 7 ; i >= 0 ; i--) {
                sendchar(pac.stream[i]);
            }

            delay_ms(23);
      }
}
//----------------------------------------------------------------------
//[Systick Interrupt Handler] SysTick interrupt happens every 0.5 ms.
void TIM1_UP_IRQHandler (void) {
      if ((TIM1->SR & 0x0001) != 0) {                 // check interrupt
source
      lCurrentTime++;
            TIM1->SR &= ~(1<<0);                          // clear UIF flag
      }
} // end TIM1_UP_IRQHandler
//-----------------------------------------------------------------------------
----------

//[Simple function to get the current time
DWORD GetCurrentTime(void) {
      return lCurrentTime;          //current time stored in increments of
0.5ms
}
//----------------------------------------------------------------------
```

HexCrawler.h Configuration File

```
//Project HexCrawler
//Description: HexCawler 1.0 configuration file.
//          All Hardware connections (excl controls) and body dimensions
//          are configurated in this file.
//Date: 19-7-2010
//Programmer: Daniel Earley
//
//Hardware setup: STM32F107VC, Parallax Servo Controller
//
#ifndef HEXCRAWLER_H_
#define HEXCRAWLER_H_

//-------------------------------------------------------------------
//[HEXCRAWLER CONFIG FILE]
#include "HexCrawler_1_cfg.h"
//#include "HexCrawler_2_cfg.h"
//#include "Nomad_cfg.h"

//-------------------------------------------------------------------
//[HEXCRAWLER LIBRARIES]

#include <stm32f10x_cl.h>                        // STM32F10x Library
Definitions
#include <stm32f10x_type.h>
#include <math.h>
#include <stdio.h>
#include "GenericTypeDefs.h"

#include "Delay.h"
#include "LED.h"
#include "ADC.h"
#include "LED.h"
#include "GPIO.h"
#include "Serial.h"
#include "ServoPacket.h"
#include "Timer.h"

//===================================================================
//[CONSTANTS]
#define BUTTON_DOWN  0
#define BUTTON_UP        1

#define c1DEC            10
#define c2DEC            100
#define c4DEC            10000
#define c6DEC            1000000

#define cRR              0
#define cRM              1
#define cRF              2
#define cLR              3
#define cLM              4
#define cLF              5
```

```
//----------------------------------------------------------------------
//[Function Prototypes]
void GaitSelect(void);
DWORD GetCurrentTime(void);
void WriteOutputs(void);
//void GPPlayer(void);
void SingleLegControl(void);
void BalCalcOneLeg(SHORT PosX, SHORT PosZ, SHORT PosY, BYTE BalLegNr);
void BalanceBody(void);
void BodyIK(SHORT PosX, SHORT PosZ, SHORT PosY, CHAR RotationY, BYTE
BodyIKLeg);
void LegIK(SHORT IKFeetPosX, SHORT IKFeetPosY, SHORT IKFeetPosZ, BYTE
LegIKLegNr);
void CheckAngles(void);
void Sound(void);
void ReadButtons(void);
void WriteOutputs(void);
void GaitSeq(void);
void ServoDriver(void);
void FreeServos(void);
void Gait(BYTE GaitCurrentLegNr);
SHORT GetAtan2(SHORT AtanX, SHORT AtanY);
void GetSinCos(SHORT AngleDeg1);
void SendPacket(BYTE Channel, WORD Position);
SHORT GetArcCos(SHORT Cos4);

#endif /* HEXCRAWLER_H_ */
```

HexCrawler_2_cfg.h Configuration file

```
#ifndef HEXCRAWLER_1_CFG_H_
#define HEXCRAWLER_1_CFG_H_

//--------------------------------------------------------------------
//[SSC PIN NUMBERS]
#define cRRCoxaPin             10   //Rear Right leg Hip Horizontal
#define cRRFemurPin      11   //Rear Right leg Hip Vertical
#define cRRTibiaPin      16   //Rear Right leg Knee

#define cRMCoxaPin             8    //Middle Right leg Hip Horizontal
#define cRMFemurPin      9    //Middle Right leg Hip Vertical
#define cRMTibiaPin      17   //Middle Right leg Knee

#define cRFCoxaPin             6    //Front Right leg Hip Horizontal
#define cRFFemurPin      7    //Front Right leg Hip Vertical
#define cRFTibiaPin      18   //Front Right leg Knee

#define cLRCoxaPin             4    //Rear Left leg Hip Horizontal
#define cLRFemurPin      5    //Rear Left leg Hip Vertical
#define cLRTibiaPin      19   //Rear Left leg Knee

#define cLMCoxaPin             2    //Middle Left leg Hip Horizontal
#define cLMFemurPin      3    //Middle Left leg Hip Vertical
#define cLMTibiaPin      22   //Middle Left leg Knee

#define cLFCoxaPin             0    //Front Left leg Hip Horizontal
#define cLFFemurPin      1    //Front Left leg Hip Vertical
#define cLFTibiaPin      26   //Front Left leg Knee

//--------------------------------------------------------------------
//[SERVO CENTER POSITION OFFSET]
#define cRRCoxaMiddle    725  //Rear Right leg Hip Horizontal
#define cRRFemurMiddle   750  //Rear Right leg Hip Vertical
#define cRRTibiaMiddle   750  //Rear Right leg Knee

#define cRMCoxaMiddle    720  //Middle Right leg Hip Horizontal
#define cRMFemurMiddle   750  //Middle Right leg Hip Vertical
#define cRMTibiaMiddle   750  //Middle Right leg Knee

#define cRFCoxaMiddle    719  //Front Right leg Hip Horizontal
#define cRFFemurMiddle   690  //Front Right leg Hip Vertical
#define cRFTibiaMiddle   750  //Front Right leg Knee

#define cLRCoxaMiddle    770  //Rear Left leg Hip Horizontal
#define cLRFemurMiddle   730  //Rear Left leg Hip Vertical
#define cLRTibiaMiddle   750  //Rear Left leg Knee

#define cLMCoxaMiddle    730  //Middle Left leg Hip Horizontal
#define cLMFemurMiddle   690  //Middle Left leg Hip Vertical
#define cLMTibiaMiddle   750  //Middle Left leg Knee
```

```
#define cLFCoxaMiddle    785  //Front Left leg Hip Horizontal
#define cLFFemurMiddle   710  //Front Left leg Hip Vertical
#define cLFTibiaMiddle   750  //Front Left leg Knee

//--------------------------------------------------------------------
//[MIN/MAX ANGLES]
#define cRRCoxaMin1            -260//Mechanical limits of the Right Rear Leg,
decimals = 1
#define cRRCoxaMax1            740
#define cRRFemurMin1     -1010
#define cRRFemurMax1     950
#define cRRTibiaMin1     -1060
#define cRRTibiaMax1     770

#define cRMCoxaMin1            -530 //Mechanical limits of the Right Middle
Leg, decimals = 1
#define cRMCoxaMax1            530
#define cRMFemurMin1     -1010
#define cRMFemurMax1     950
#define cRMTibiaMin1     -1060
#define cRMTibiaMax1     770

#define cRFCoxaMin1            -580 //Mechanical limits of the Right Front
Leg, decimals = 1
#define cRFCoxaMax1            740
#define cRFFemurMin1     -1010
#define cRFFemurMax1     950
#define cRFTibiaMin1     -1060
#define cRFTibiaMax1     770

#define cLRCoxaMin1            -740 //Mechanical limits of the Left Rear Leg,
decimals = 1
#define cLRCoxaMax1            260
#define cLRFemurMin1     -950
#define cLRFemurMax1     1010
#define cLRTibiaMin1     -770
#define cLRTibiaMax1     1060

#define cLMCoxaMin1            -530 //Mechanical limits of the Left Middle
Leg, decimals = 1
#define cLMCoxaMax1            530
#define cLMFemurMin1     -950
#define cLMFemurMax1     1010
#define cLMTibiaMin1     -770
#define cLMTibiaMax1     1060

#define cLFCoxaMin1            -740//Mechanical limits of the Left Front Leg,
decimals = 1
#define cLFCoxaMax1            580
#define cLFFemurMin1     -950
#define cLFFemurMax1     1010
#define cLFTibiaMin1     -770
#define cLFTibiaMax1     1060
//--------------------------------------------------------------------
//[BODY DIMENSIONS]
#define cCoxaLength      29         //Length of the Coxa [mm]
```

```
#define cFemurLength      76          //Length of the Femur [mm]
#define cTibiaLength      106  //Lenght of the Tibia [mm]

#define cRRCoxaAngle1     -600 //Default Coxa setup angle, decimals = 1
#define cRMCoxaAngle1     0           //Default Coxa setup angle, decimals = 1
#define cRFCoxaAngle1     600  //Default Coxa setup angle, decimals = 1
#define cLRCoxaAngle1     -600 //Default Coxa setup angle, decimals = 1
#define cLMCoxaAngle1     0           //Default Coxa setup angle, decimals = 1
#define cLFCoxaAngle1     600  //Default Coxa setup angle, decimals = 1

#define cRROffsetX              -43  //Distance X from center of the body to
the Right Rear coxa
#define cRROffsetZ              82          //Distance Z from center of the
body to the Right Rear coxa
#define cRMOffsetX              -63  //Distance X from center of the body to
the Right Middle coxa
#define cRMOffsetZ              0           //Distance Z from center of the
body to the Right Middle coxa
#define cRFOffsetX              -43  //Distance X from center of the body to
the Right Front coxa
#define cRFOffsetZ              -82  //Distance Z from center of the body to
the Right Front coxa

#define cLROffsetX              43          //Distance X from center of the
body to the Left Rear coxa
#define cLROffsetZ              82          //Distance Z from center of the
body to the Left Rear coxa
#define cLMOffsetX              63          //Distance X from center of the
body to the Left Middle coxa
#define cLMOffsetZ              0           //Distance Z from center of the
body to the Left Middle coxa
#define cLFOffsetX              43          //Distance X from center of the
body to the Left Front coxa
#define cLFOffsetZ              -82  //Distance Z from center of the body to
the Left Front coxa
//--------------------------------------------------------------------
//[START POSITIONS FEET]
#define cRRInitPosX       53          //Start positions of the Right Rear leg
#define cRRInitPosY       25
#define cRRInitPosZ       91

#define cRMInitPosX       105  //Start positions of the Right Middle leg
#define cRMInitPosY       25
#define cRMInitPosZ       0

#define cRFInitPosX       53          //Start positions of the Right Front leg
#define cRFInitPosY       25
#define cRFInitPosZ       -91

#define cLRInitPosX       53          //Start positions of the Left Rear leg
#define cLRInitPosY       25
#define cLRInitPosZ       91

#define cLMInitPosX       105  //Start positions of the Left Middle leg
#define cLMInitPosY       25
#define cLMInitPosZ       0
```

```
#define cLFInitPosX      53         //Start positions of the Left Front leg
#define cLFInitPosY      25
#define cLFInitPosZ      -91
//---------------------------------------------------------------------

#endif /* HEXCRAWLER_1_CFG_H_ */
```