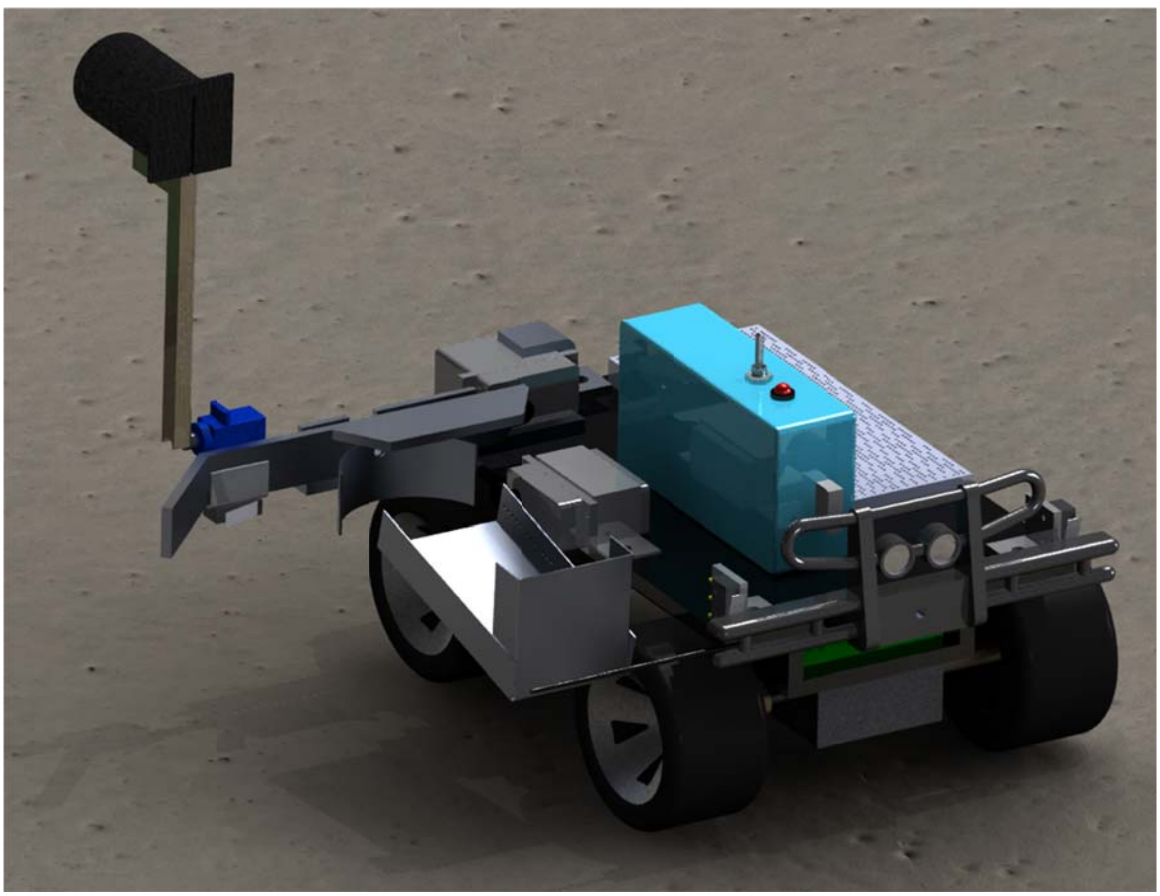
December 9, 2009

MME 4487A

2009

Autonomous Mining Robot



r

Contents

List of Figures

Introduction

Mechatronic systems integrate mechanical, electrical and software subsystems to create more flexible highly versatile machines. This course MME 4487A demonstrates how these three different fields of study can be incorporated into a concurrent design which accomplishes a predetermined task autonomously. The goal of this project was to design, construct and most importantly test an autonomous mining robot. The autonomous robot was to navigate a course with three infrared towers seeking out crystals and collecting as many as possible, and then deposit the collected crystals within one of two different receptacles within the competition time limit. Our team decided design a four-wheel drive robot which was very simple and could be completed with ample testing time to perfect the subsystems. The following report outlines the development of our autonomous mining robot and its subsystems.

Discussion

Methodology

Opening Sequence

To meet the 25cm cubic constraints of the project, the robot was fitted with three servos which compressed the size of the robot. Once the power was turned on, the hook mechanism swings out to the side of the robot, the sensor adjusts to appropriate height facing forward and the scoop moves to the driving position.

Course Orientation

The robot uses an infrared sensor which is mounted on the pocket of the hook mechanism. It uses this sensor to determine which direction to drive. The robot after sensing the correct tower drives straight for 2 seconds and the checks for the tower again. This continues until the bumper switch on the hook is tagged, telling us that we have reached the tower.

Picking up Crystals

Once the robot reaches the pickup tower it moves the scoop to the pickup angle and drives a full circle around the tower using the hook to guide it. After one full circle, the scoop is moved to a carry position and the robot reverses to detach from the tower. From here it uses the infrared sensor to drive itself to the drop off tower. On its way to the drop off tower, the robot can end up with the pickup tower in its path. A front bumper is used to detect this occurrence and if it is hit, it drives itself around this tower.

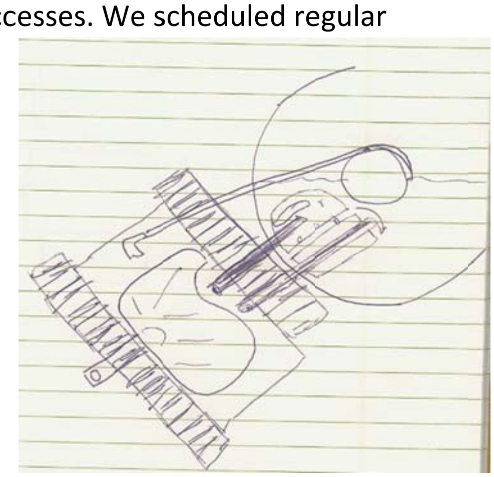
Dropping of Crystals

The robot uses the hook bumper again to determine when it has reached the drop off tower. Once this has been hit, the scoop moves to the drop off position and the crystals are dumped onto the surface of the container. It then drives a full circle around the tower and pushes the crystals around until they reach the deposit hole, where they fall into the container.

Concept Generation

Scoop & Pick-Up Concepts

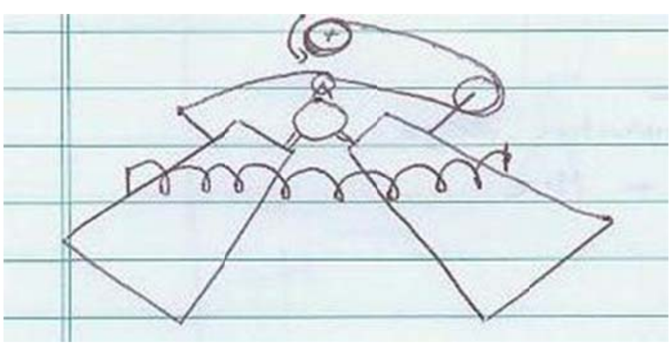
To begin this project we formed our dynamic and innovative team, comprising of one electrical student and two mechanical students with everyone having a variety of skills that would complement the team's future successes. We scheduled regular meeting times throughout the term in order to meet the milestones drafted up for our first design review. At our first meeting after the laboratory exercises were completed we discussed the strategy we would use to seek, collect and deposit the crystals. The laboratory exercises showed us how to apply coding and



circuitry to various components (servos, twin DC motor gearbox, sensors…etc) all of which we would eventually utilize in our robot design. With these components in mind and our basic knowledge of the subsystems required for a mechatronic machine such as an autonomous robot, we brainstormed several unique concepts.

We decided our robot will use a hook to wrap around the tower and collect the crystals which then deposits into the more difficult, small-hole receptacle. That concept is outlined by a sketch made in our design notebooks, see Figure 1. The hook would be critical in both the pick-up and drop off of the marbles. The figure shows the vehicle docked to the tower via the arm as a scoop is picking up marbles from the side as it circle the tower. This is how we would collect the marbles since they are located at the base of the tower but only on one side of the tower. The robot would dock to the tower and just circle the tower while scooping, hence getting a large amount of the marble around the base of the tower. As can be seen in this early sketch at the drop of area the same hook would be deployed and latched around the tower allowing the vehicle to drop the crystals onto the pedestal and then drive around using a sweeping mechanism to push all of the crystal into the hole. Designing for simplicity, this particular device would require only one servo to actuate the arm into the correct position. The

Since we had established that we would be scooping from the side when we utilize the hook mechanism to dock with the various towers we began to brainstorm the scoop concepts. The first scoop design was a claw type design operated with a motor



and closed with a return spring. Figure 2, shows this concept from a sketch out of the design notebook and illustrates this concept quite well. A claw type of scoop would

ensure that the crystals do not fall

out and more importantly have a high chance of getting large amounts of crystals. The downfall of this idea is the complexity of the mechanism which opens and closes the scoop, since it may be too difficult to properly execute. Although it resembles the claw commonly seen on most industrial excavators, the crystals in the environment we are planning to use it in are too lightweight. The crystals are thus displaced rather than plowed and shoveled into the buckets.

Figure 3, Illustrates another design concept for the scooping mechanism of the robot. This design uses a stationary bucket which is lowered into the pick-up tray where the crystal are located and then the as the robot drives along the scoop allows a large amount of crystals to be stored inside the bucket. When the robot stops the front of the scoop swings down to collect the crystals that have accumulated at the leading edge of the stationary bucket. The closed scoop has now sealed the bucket and allows the scoop to be lifted and the crystal would be placed into a receptacle on-board the deck of the robot (basin).

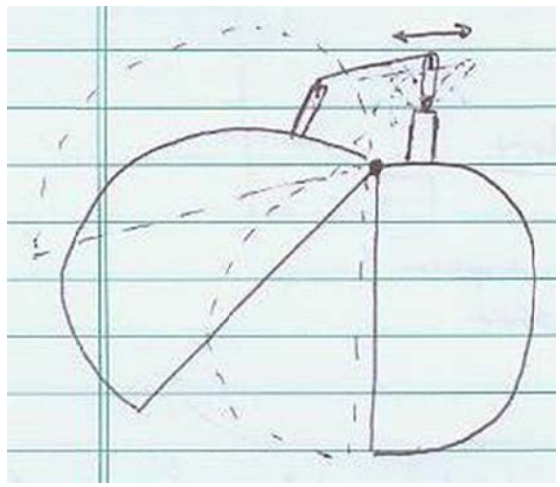
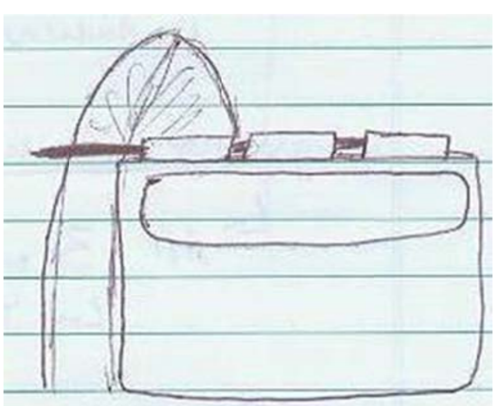


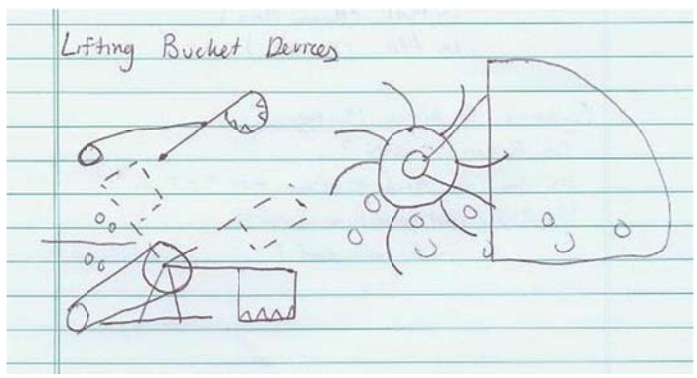
Figure 4, shows how there would be a space in the top of the scoop that when the closed bucket is placed upside down the crystal can freely fall out into the basin. There could be a slight lip on the back side of the scoop to help guide the crystals into the on-board receptacle. This design was our favorite concept since it seemed very simple to actuate and would be able to hold high volume of crystals. However the problem that could be detrimental to this type of the design is when



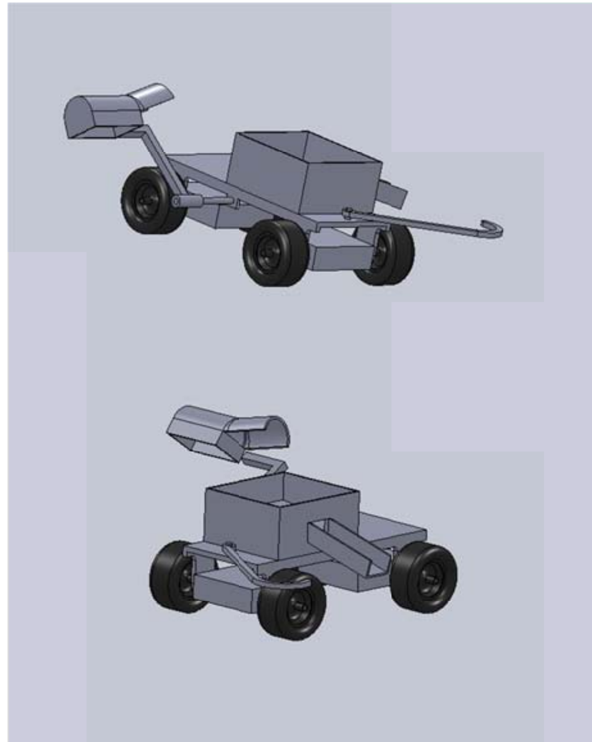
Exit Slot

the scoop deploys and begins the pick-up run it may not displace the crystals below the scoop. Therefore if the scooped was lowered over top of the crystals it would not gather any into the bucket since the scoop and servo are not heavy or powerful enough to dig through the crystal pile. Another problem with the concept is the need for an onboard receptacle, which complicates the robot and increases failure modes when compared to a more simplistic design.

The most complex but possibly most effect concept that was generated is shown in Figure 5. The scoop is somewhat similar to the bucket-design explained above; therefore it also may exhibit some of the downfalls as the Two-Part Scoop. As shown in Figure 5, the scoop has a stationary bucket and instead of using a frontal scoop to aid in the collection of crystals there is a paddle wheel or brush. This may decrease the inability to collect crystals by plunging through the large pile of crystals with the front of the paddles/brushes as the robot circles the tower. Also shown in the picture is a simple motor and belt drive system used to lift the bucket after the pick-up run is completed. This high torque task would be well suited for stepper motor but due to belt slippage and complication in the lifting of the scoop this concept was also deemed too complex for our application. Also this design requires the on-bard basin to be installed which much again be emptied by another subsystem at the drop off location. By removing the need for an on-board receptacle we reduce the number of parts, actuators, and decrease complexity of the code. Also another major downside to having an onboard receptacle is that you must be able to have the hook on both sides of the robot, because there is likely not enough space-claim on the bucket side of the robot. Thus, entailing a high amount of complexity in the layout and manufacturing of this efficient but incredibly difficult scooping mechanism.



The next figure shows the first 3D CAD model of our robot constructed using a SolidWorks (Figure 6). At first we planned on using the swiveling bucket design shown in the CAD model, however through our failure mode analysis it was deemed too complex.

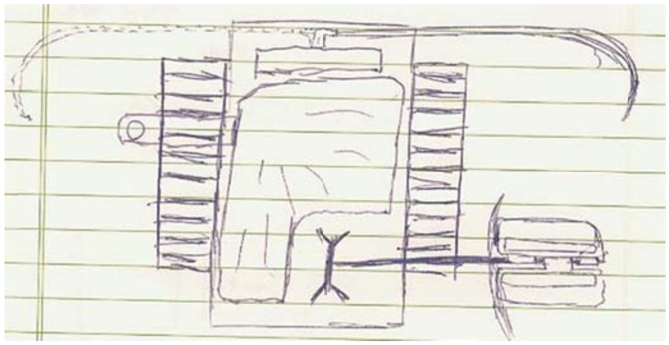


op &

The design was unlikely to be as successful as something a lot simpler due to the non-uniform surface at the crystal pick-up area. As explained above the receptacle on-board the robot creates more complexity for our design goals of picking-up the crystals on one side of the robot. The receptacle occupies a large amount of the upper deck of the robot on this preliminary chassis design. Since gravity is the method for which the crystals would descend from the basin into the small hole at the drop-off area that height difference and funnel that brings the crystals onto the depositing chute reduces the amount of space we have to mount other sensors and devices. This problem can develop into a devastating issue because there a strict 25cmx25cmx25cm size restraints.

Drive Train Concepts

The following figure shows an image taken from one of our design notebooks outlining some of the concepts that were developed by sketching and brainstorming. This sketch shows the chassis set-up for our initial concept which uses two “tank-like” tracks to navigate eh course. Tracks initially seemed like the best option, however upon further review we found that since the robot would need to spin on the spot until it locates and hones in on a particular tower. This spinning on the spot with tracks could create a large gully or grove in the aggregate rocks that make up the surface of the course. During the first design review we meet with both the professor and several teaching assistants which informed us in previous years there have been a great deal of



failures due to tracked systems. Though discovering that a four-wheel drive system would excel whilst navigating the gravel aggregate we shifted focus and decide to implement the new drive train goal in the new concepts.

Programming and Testing

Once the design concept was established, and a model chassis was constructed, the next step was to begin programming and testing the robot. The programming was to be completed using MPLAB and done in machine language. During the first month of the semester, there were four laboratories assigned for ECE 4487a students. These four labs focused on understanding basic programming concepts and introduced strategies used to control components that may be used on the final robot, such as DC, servo and stepping motors, as well as light and infrared sensors. These concepts were a basis for the beginning stages of programming our robot.

The programming structure was broken down into individual subroutines. This was done with intent to separate the overall program into many smaller programs which could be tested on their own. The following sequence was followed while programming:

1. Global Time Delay
2. Drive forward, reverse and spin
3. Bumper Switch
4. Tower Detect
5. Servo
6. Pickup
7. Dropoff
8. Drive Around Tower

Before each major subroutine was written a flowchart was completed. The flowchart was used to layout the structure of the subroutine itself. This provides a basis to start coding and ending point. Flowcharts are attached in Appendix A. The program code can be found in Appendix B.

Global Time Delay

The purpose of this routine was to establish a global delay subroutine that could be used throughout the program. The direction taken initially was to try and utalize a method stated in lectures using internal timer TMR0. This posed an issue considering we planned to use the same Timer (TMR0) to code our servo motors as done within LAB 3. This idea was scrapped and nested for loops were used to create a delay. Three loops were required to get a delay of at least 5 seconds. The subroutine was constructed with the ability to define a variable that was used to send the desired time delay when calling the subroutine. This variable was then used in the outermost loop which in turn caused a delay.

Drive Forward, Reverse and Spin

This routine was created with intent to test the chassis that was constructed. The basic foundation of the routine was created in Program 6 of LAB 3. The program was modified to adjust speed of the DC motors using a defined variable instead of theanalog to digital conveter which was used in the lab. It uses the time delay subroutine that was created to drive for an set amount of time. This code provided the necessary trials required to determine, direction of DC motors, as well as desired speeds which could navigate the rough terrain of the course. This routine is used to drive straight toward the tower.

Bumper Switch

This routine was assembled to test the capability of the bumper switch. A test was completed to see whether we could drive straight into a tower, tag the switch and reverse from the tower. There were two options to choose from when coding the switch. Either use an external interupt method, which put priority on the switch itself over any of the main body code. This was our first direction, but the fact that the microcontroler that was to be used, only contained one external interrupt. This interupt was planned to be used for the infrared sensing of the towers. Considered using the same external interrupt for both, using flip flops to determine which device caused the interupt. This added additional difficulty and was decided against using the method. The second approach was to use polling of the bumper switch input to continually check if it had been tagged. Using this method required that the polling be done in a part of the code that constantly was used. Potentially the switch could be tagged at a point where the polling was not being completed and could be missed entirely.

An issue arose in initial testing, where the robot would reverse without the switch being tagged. It was determined that the problem was with the circuit that was set up for the bumper switch itself. Initially we had the switch wired up to 5V and checked if the input to the microcontroller was “high”, meaning the switch was tagged. This was an issue because when the switch was not tagged, the input pin was floating. Any noise withing the circuit could cause a floating pin to increase in voltage to a point where the microcontroller believes that the pin saw a “high” thinking that the switch had been depressed. This was resolve by wiring the switch to ground,so that the controller read “high” until the switch was depressed where it saw “low”. See circuit diagram below.

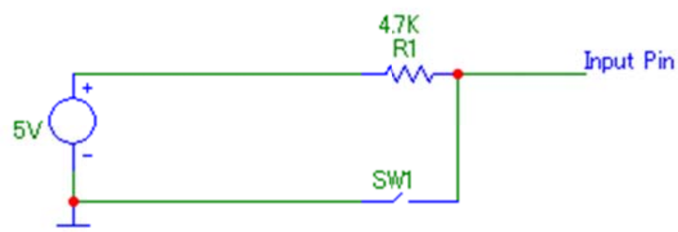


Figure 8: Bumper switch

Tower Detect

This piece of the code is used to detect which tower is being sensed. It is an integral part of the overall program, because it is used to determine the direction in which the robot needs to drive. This routine was integrated with the Driving routine to test functionality. Again the foundation of the program was obtained from LAB4. The infrared sensor uses Timer 1 to determine the period of the signal being received, which gets compared within the program to determine which tower is being sensed.

Initially the integration of the program did not provide expected results. The towers appeared to have been detected, since that appropriate LED's were lighting up as in LAB4, but the robot would not drive toward the appropriate tower. After using the debug tool within MPLAB it was found that the "BeaconCheck" variable was not returning the appropriate value when checking which tower was being sensed. This was corrected by removing this variable and directly comparing the timer value within the main body of the program. This fixed the issue.

Servo

The last devices to be integrated into the program were the servo motors that were planned to be used for the hook mechanism and the scoop. The base again was used from a previos lab, and was modified to use a determined value to set the angle of the servo. The motors are sent the desired angle signals unsing the internal Timer 0 (TMR0). Issues arose when initializing the interrupts. Since we were integrating both Timer1 and Timer 0 into the same program, we needed to make sure all the appropriate intterupts were enabled and that the priorities within the interupt subroutine were correct.

Pickup and Drop Off

Once all the devices were integrated within one common program, the next step was to begin picking up crystals using all the components. The pickup and drop off programs are similar instructure. We use the bumper switch located on the hooking mechanism, we know when we have reached the tower. From there the pickup and drop off routines are hardcoded to perform a specific sequence of events defined seperatly. Since the code written for this did not add any new components, the only tweaking that occurred was with the amount of time to drive around the towers, and the angle at which the servos are set at.

Drive Around Tower

This routine was added after final trials. It is hardcoded to manouver the robot around a tower if it detects that the front bumper switches have be tagged. Again the only tweaking that was completed was direction of the DC motors and the angle of the servos

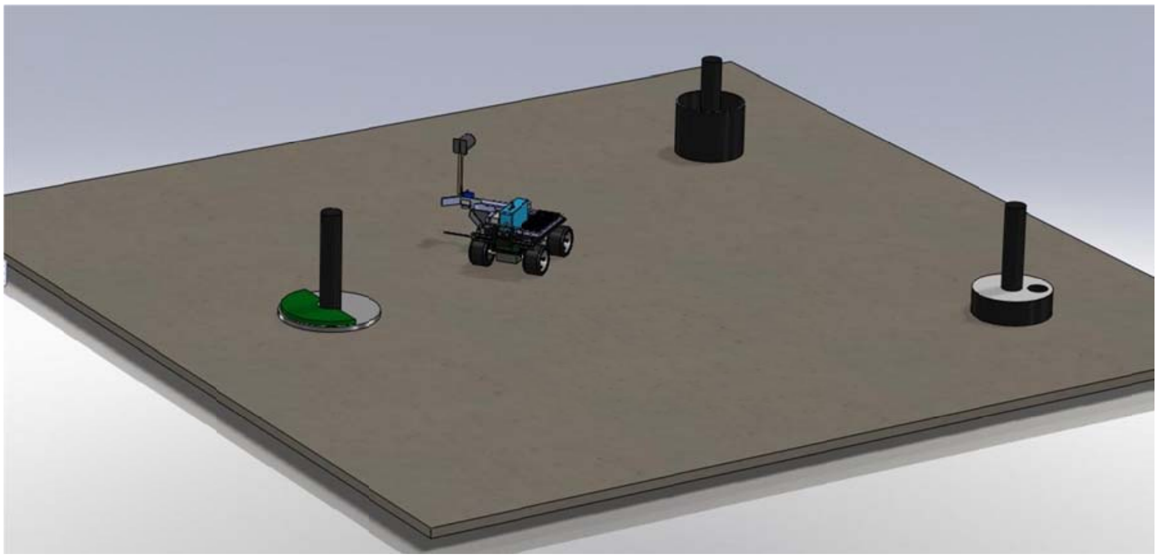
Troubleshooting and Debugging

Problem Encountered	Solution
Beacon sensor detecting multiple towers	Tube casing was constructed, which was fitted with several painted black deflectors were inserted into the tube, leaving a small line which narrowed the vision
Confusion when wires popped out of breadboard due to similar coloured wires used	Labbelled important wires with pin numbers to avoid confusion
Noise within the circuit causing improper movement of servos, DC motors, sensing capabilities and bumper switches	Added filter capacitors to clean the up the signals and insulated cable for the bumper switches
DC motor on one side was starting to degrade and moving at a different speed than the other side	Adjusted speed of motor within the program to account for this discrepancy
Origional placement of sensor drove the robot directly into the tower. Needed to hook onto the tower	Moved Sensor placement to above the hook so that the robot drove the tower into the pocket of the hook
When driving straight initially the sensor would lose the tower and spin a full 360 degrees to find the tower again	Added a slight turn to the left while driving straight so that we would lose the tower on the same side everytime and not have to spin as long to regain the signal
During intial design trials, it was noticed that after picking up crystals if the robot ended with the pickup tower between it and the drop off tower that it would be stuck behind it	Added a front bumper to detect if we get stuck and once tagged it drives around the tower it is stuck behind
Initial design did not fit in the cubic restraints	Added an extra servo to control the sensor, so that it would bend in and fit in the 25 cubic cm box
Pushing arm was was mounted too high and crystals were sliding underneath and not being pushed around the drop off tower	Since the length of this plate was restricted due to the scoop, duct tape was added to get the length required but was easily displaced by the scoop when in drive position.

Final Design

Course

The following Figure 8 shows an image taken of the virtual course that we constructed using a SolidWorks.



Aggregate

- Approximately ½ inch of gravel pebbles spread out uniformly over the surface of the table

Beacon Towers

- Three towers transmitting three uniquely different frequencies of Infra-red (IR)
- The tower showing the green semi-circle indicates the pick-up area of the course
- The tower in the bottom right-hand side of the image illustrates the more challenging "small-hole" drop-off location,
- The remaining tower is the larger drop-off receptacle, is located by using the 170Hz signal
- The towers were manufactured by the UWO electronics shop using an array of 12 (twelve) or more IR LED's arranged in a array dispersing the resulting signal radially outwards away from the tower

The towers emit frequencies of 170Hz, 85Hz, 340Hz, using a carrier frequency of 38 kHz

Crystals

- The crystals/gems are located in the green shaded are of Figure 8, located inside a silver dish(plate)

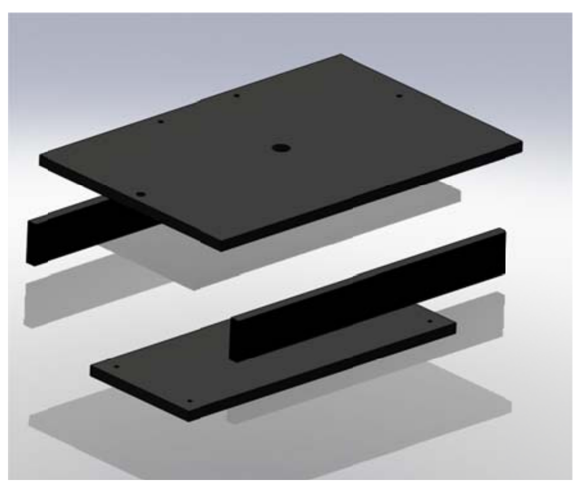
Robot Components

Mechanical/Electrical

The goal of the autonomous robot design was to be holistically simple system, which naturally allows for each of the subsystems to be as simplistic as possible. A simpler the robot design allows for many advantages compared to a more complex system, some of which include:

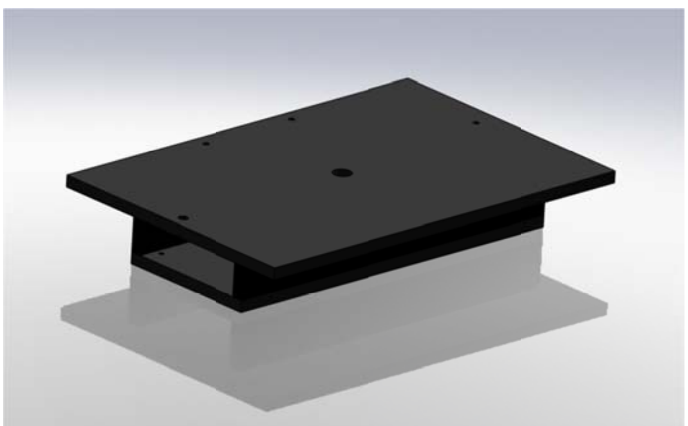
- reliable system
- ease of manufacture/fabrication
- low part count
- easier to troubleshoot/debug potential issues

These are all very important to the success of the robot during the competition due to the very strict constraints and difficult conditions of the course. Maintaining simplicity throughout the design of the mechanical structure and minimizing the use of complicated electrical circuitry is the core to the teams' success.



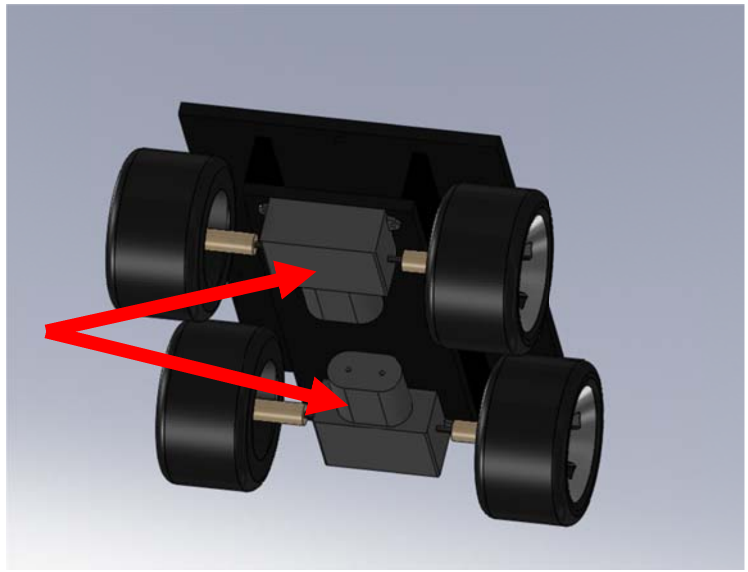
Chassis

The chassis is made out of medium density fiber (MDF) board. Initially aluminum was considered for the core structure; however since the supplied DC motors are quite weak, the lighter more versatile MDF structure was fabricated. The MDF was bonded together using a high strength phenolic-based epoxy, which created bonds stronger then the individual boards themselves. The strongest part of the MDF structure is the skin because the center of the fiberboard is very "crumbly" and weak. Since we might be screwing fasteners into the material with woodscrews for added reinforcement the final chassis was coated in the high strength phenolic resin and then heated to allow the MDF to soak up the resin. This process was repeated approximately three times until the inner structure was soaked in epoxy creating a very lightweight yet high strength chassis. The MDF structure allowed for a simple expedited manufacturing and created a part that could be screwed into, bolted to, or glued to anywhere without the need to pre drill holes. The following figure, Figure 9, shows an exploded view of the pieces prior to gluing and then Figure 10 illustrates the chassis after bonding, with a clearance hole for the wires drilled for the DC motor drivers. The figures were created using SolidWorks and the drawings used to fabricate the pieces can be found in the Appendix H. The chassis was then painted black to minimize reflections and be aesthetically pleasing.



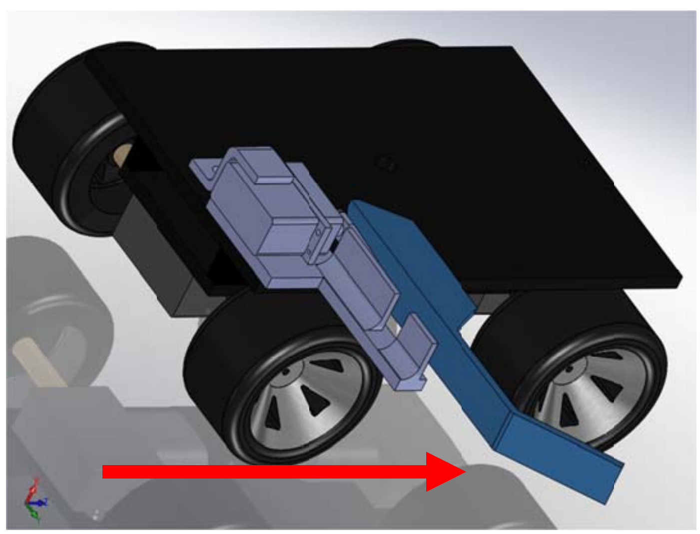
Mobility System

A four-wheel drive system was chosen to be the best suited drive train for the completion environment. The four-wheel drive system functions effectively in all aspects of navigating the course and can be driven quickly across the aggregate. The team purchased a Hummer H3T RC car and stripped the vehicle for components. The wheels were used as part of our robot to provide ample traction and an interface to which we mounted a drive-shaft extension. The four wheels were coupled with two TAMIYA Double Gearbox set to the 344.2:1 gear ratio setting. Each gearbox contained two DC motors wired in series to the opposing gear box. As seen in Figure 11, the twin motor gearboxes are mounted opposing each other and the wheels are attached via the shaft extensions. This drive train package was very simple to wire, program and attach to our chassis therefore we met our design goals and the setup worked very effectively.



eels

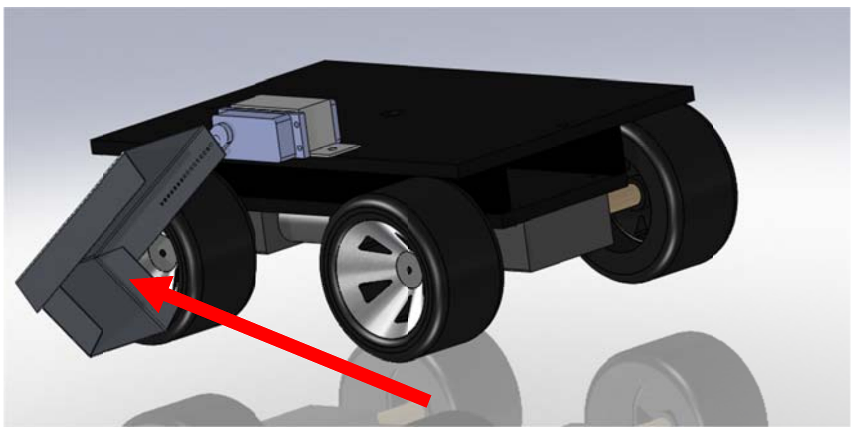
Docking Arm



The docking arm is a dual purpose hook mounted on a servo used for pick-up and drop-off of the crystals. The arm is lowered from its home position (vertical) to the horizontal functional position and rests on a supportive bracket. The supportive bracket keeps the docking arm in a horizontal position but also prevent the arm from deflecting backwards when docked with the beacon tower. The arm also serves as a mount for the IR sensor subsystem, sweeper and the micro switch.

Scoop

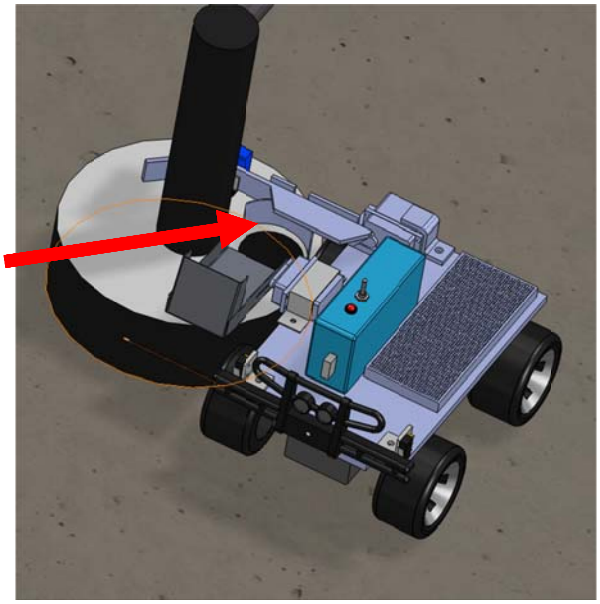
The scoop was fabricated out of sheet aluminum and designed to be as simplistic as possible. The scoop is critical in placement due to the very strict size constraints dictated by the competition. The scoop may seem unsophisticated but was highly effective at maximizing its capacity because the very thing sheet aluminum cut into the pile of crystals as it deployed rather than pushing them aside. The scoop was mounted on a servo facing the beacon tower and would start the mining run retracted towards the rear of the vehicle. Then as the pick-up routine was initiated immediately after



docking the scoop was lowered and beginning to excavate crystals. After completion of the pick-up routine the scoop is raised partway to prevent crystal from falling while traveling to the drop-off receptacle. Once the drop-off had been initiated the scoop would be lifted and the crystals drop onto the receptacle, later swept in by the robot sweeper attachment.

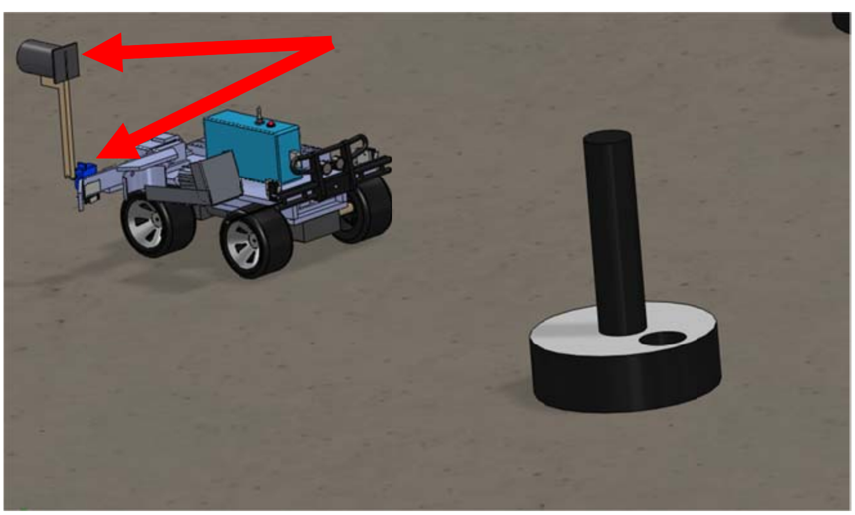
Sweeper

The sweeper unit is mounted onto the docking arm and provides a cradle into which the crystals are captured after they have been dumped at the drop-off location. Then as the vehicle moves around the receptacle the sweeper acts as guide funneling all the remaining crystal which didn't fall directly into the hole.



Beacon Sensor/Blinder Unit

The beacon sensor is a TSOP1238 light senor which detects a given set of IR frequencies. It is designed to detect signals with a carrier frequency of 38 kHz and is



amplified directly, then processed in the microchip. The TSOP1230 is blinded by a unit made out of carbon tubing with baffling painted matte black to give a focused non-reflective blinder. This allows the robot to align itself with the tower by only recognizing the beacon towers pulse when aligned with it through the front of the slit in the blinder. The blinder unit which houses the TSOP1238 is mounted on a servo that is attached to the docking arm. When the docking arm is lowered to the horizontal at the start of the program the blinder unit's arm erects into a vertical position in order to detect the towers.

Bumper

During testing, it was identified that there were some starting locations on the course that caused the robot to drive into the pickup tower while making its way to the drop-off location. After considering software modifications it was determined that the most reliable way to overcome this issue was to install a bumper on the front that triggers a redirect function to allow the robot to reorient itself. The bumper provided in the New Bright Hummer H3 RC car kit was well suited for the application with minor modifications so it was eventually used. The figure below shows the modification made which includes a functional but not aesthetically displeasing coat hanger.

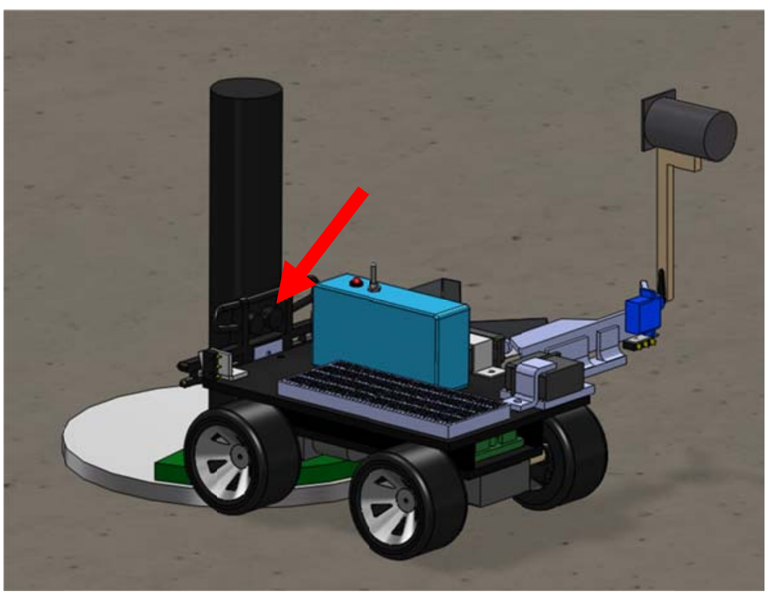


Figure 17: Bumper being used to trigger robot redirection

Figure 18: Bumper modified with coat hanger extension

RC Servo Motors

Three servo motors were used on the robot. One extends or folds the optical sensor, one controls the scoop, and one to lift or lower the docking arm. The servo that folds the optical sensor is a lightweight mini servo and the others are regular RC servos.

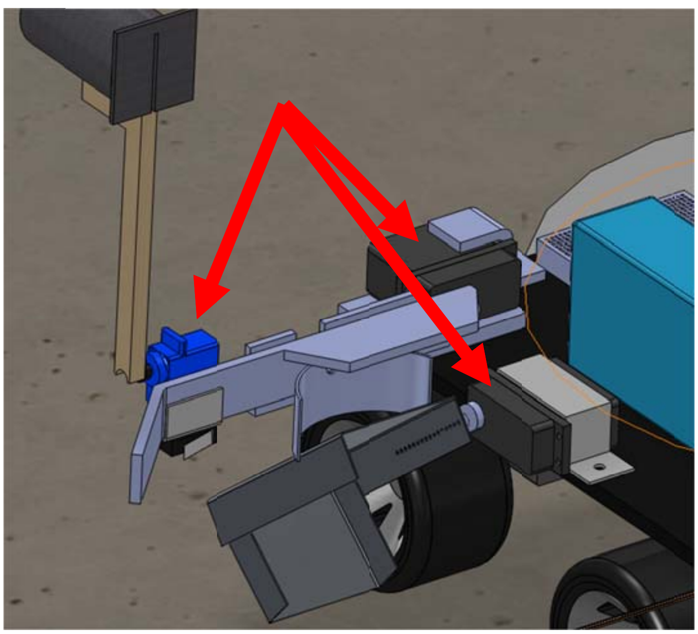


Figure 19: RC servos used in robot

Dimensions of the two sizes of RC servos may be found in Appendix H

Breadboard

Wiring was completed on the breadboard provided in the labs. Testing verified that it was reliable enough for the purposes of our prototype and using a breadboard rather than a printed circuit board allowed modifications to require little effort.

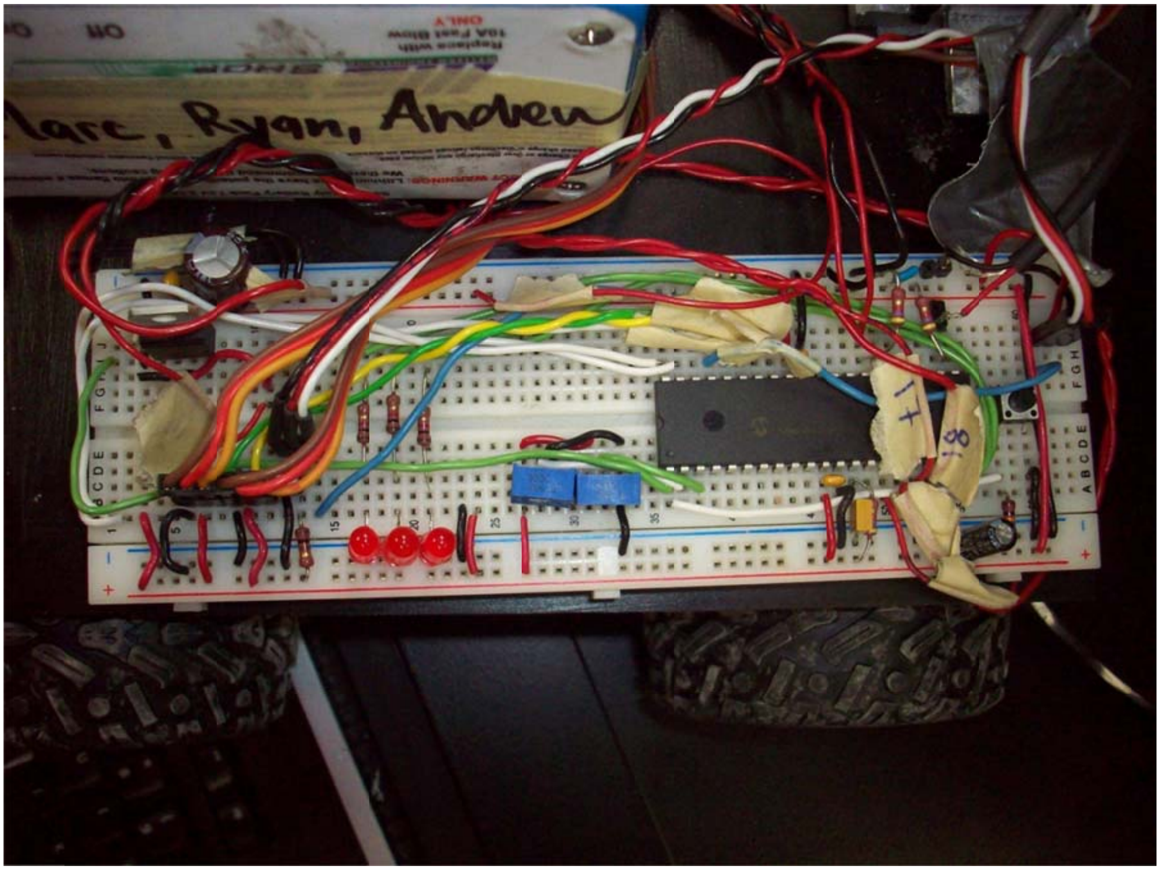


Figure 20: Breadboard

Motor Drivers

Two motor driver boards were used on the robot. One drove the left wheels of the robot and one drove the right wheels. The motor drivers used were designed and built by the UWO Electronics Shop. The circuit within the driver uses the MC33887 H-bridge and features load current feedback although the feedback functionality was not used in the robot prototype.

The robot drivers were mounted on the lower deck of the robot with wires going up through a hole to the breadboard and down through a hole to the DC motors. The location may be seen in the below figure.

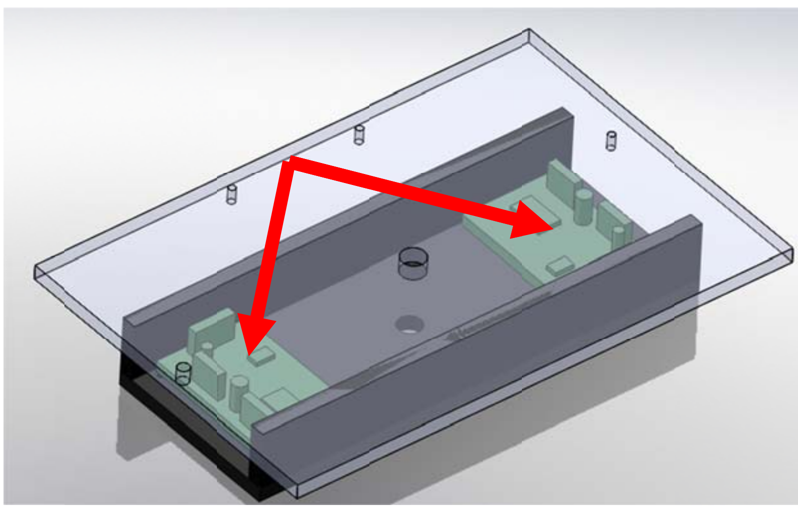


Figure 21: DC motor drivers in robot chassis

Micro Switch

Three microswitches were used in the robot prototype. Two were used on the bumper and one was used on the docking arm in order to detect the towers. The microswitch operates by applying pressure on the lever arm which depresses a momentary switch.

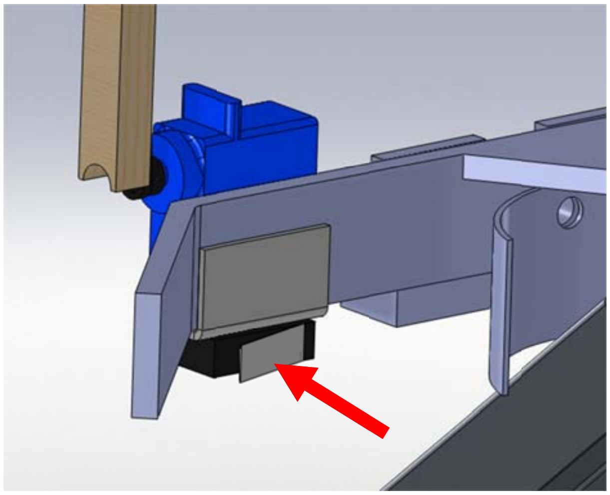


Figure 22: Microswitch used on docking arm

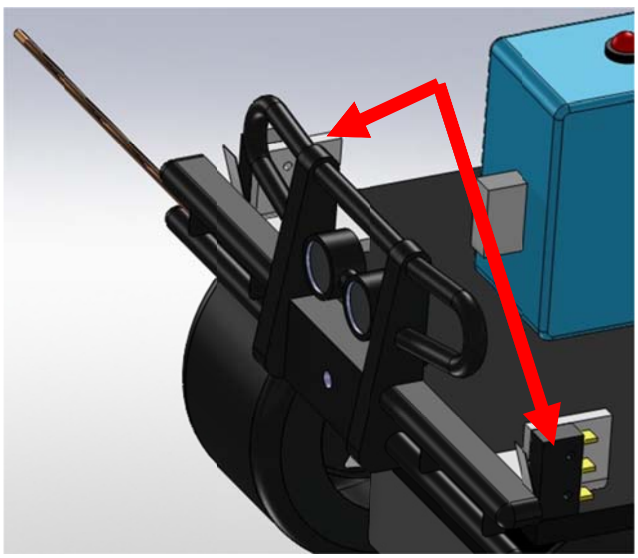


Figure 23: Microswitches used on front bumper

Dimensions of the microswitch may be found in Appendix H

Wiring

Careful attention was paid to organizing the wires neatly on the board to allow for simple troubleshooting and reduce the likeliness of wires inadvertently being pulled out.

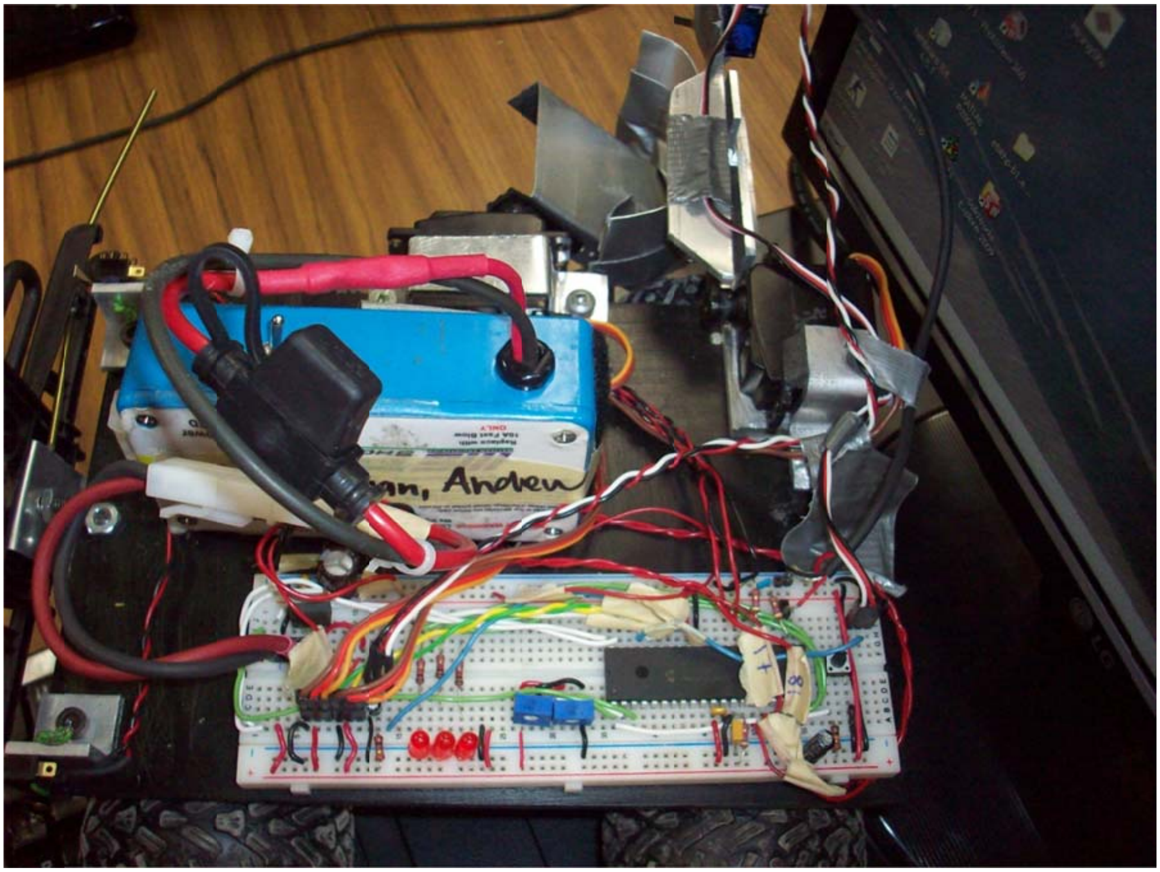


Figure 24: Wiring

All wires terminating on motors or switches were soldered and insulated with heat shrink to reduce the likelihood of an unwanted short circuit.

Battery

The battery used was an 8.3 volt pack designed and built by the UWO Electronics Shop. The capacity of the battery is 2.1 amp-hours. A picture of the battery may be seen below:

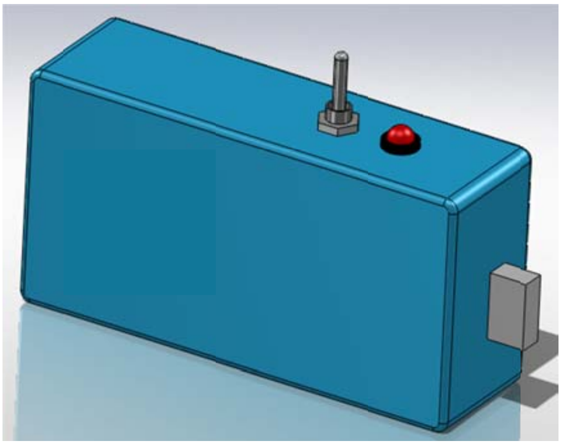


Figure 25: Battery

Dimensions of the battery may be found in Appendix H.

Conclusions

Our team was able to design, build and compete with our autonomous mining robot. We strived to make our design as simple as possible which was accomplished by many iterations eventually leading to our final design. Our robot was the first to complete the course without any assistance during the testing phase and consistently got approximately six to eight crystals. We thereby accomplished our goals by finishing early with a simple robot having time to spare however during the competition several factors contributed to a less than average performance. Nevertheless we placed third overall in the competition and achieved the most consistent robot of the term. Everyone on the team has learned a great deal through this project and been able to apply the engineering design methodology to a real-life application. This was a rewarding and successful project that was not only a great learning experience but more importantly a first step towards real-life problem solving and engineering innovation.

Recommendations & Design improvements

The final design of our robot consisted of most of our original design concepts. It performed the task autonomously, but was just shy of the time constraint during the competition. A few recommendations can be made to improve upon the design.

To address the time constraint issue that was realized, this can be attributed to two full spins when trying to find the correct tower. The two spins lost a good 40 second from the beginning. These extra spins could have been avoided if the infrared sensor was mounted to a point on the robot that did not shake. The servo that was added to meet the size constraints, added more potential for error due to the vibration of the sensor. Another approach to resolve the time constraint issue would be to speed up the DC motors during the spin routine. We had previously tried to speed up the spin routine, which cause the sensor to miss the signal because it was moving too fast.

A second improvement that could be made would be to mount the hook bumper switch more towards the pocket of the hook and extend the switch arm so that it gets tagged as soon as a tower is hooked onto. In the second run of the competition we experienced an issue where the hook bumper switch did not get tagged as soon as it hooked onto the tower. This issue cost us time at the pickup and drop off points.

Another improvement could be modifying the gear ratio of the DC motors. The configuration used was the highest torque possible. This gear ratio causes a sacrifice in operational speed. If we used a different ratio with less torque we may have been able to perform multiple runs and collect more crystals.

An electrical enhancement to the robot would have been the addition of a printed circuit board. The addition of this board would remove any possibility of wires popping out of the bread board during trials and the competition. This would also provide a more professional clean look to the robot.

The programming could have been improved by hard coding routines less and creating more global routines that could be used throughout the program. This would shorten the length of the overall program and provide easier troubleshooting. This was attempted but due to time constraint near the end of the semester, it did not reach full potential.

Outside of these major recommendation, minor improvements such as adjustable components, such as the sensor mount, location of push plate as well as scoop. The size of components could have improved, including the scoop size to pick up more crystals, which in turn would require a smaller frame size.   These small design changes could have produced a more efficient robot but due to the time constraints were not addressed.

References

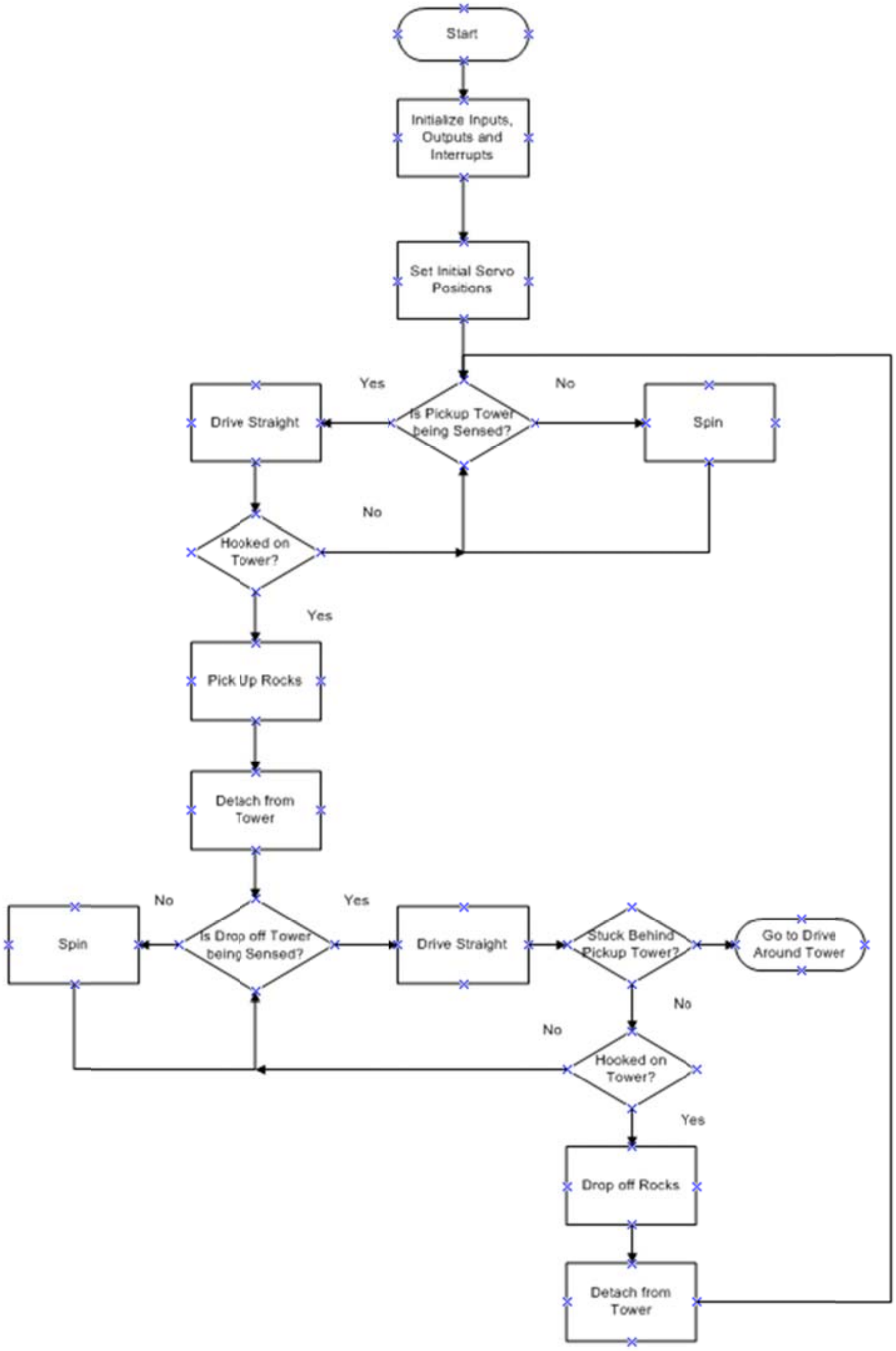
MME 4487A Lecture notes

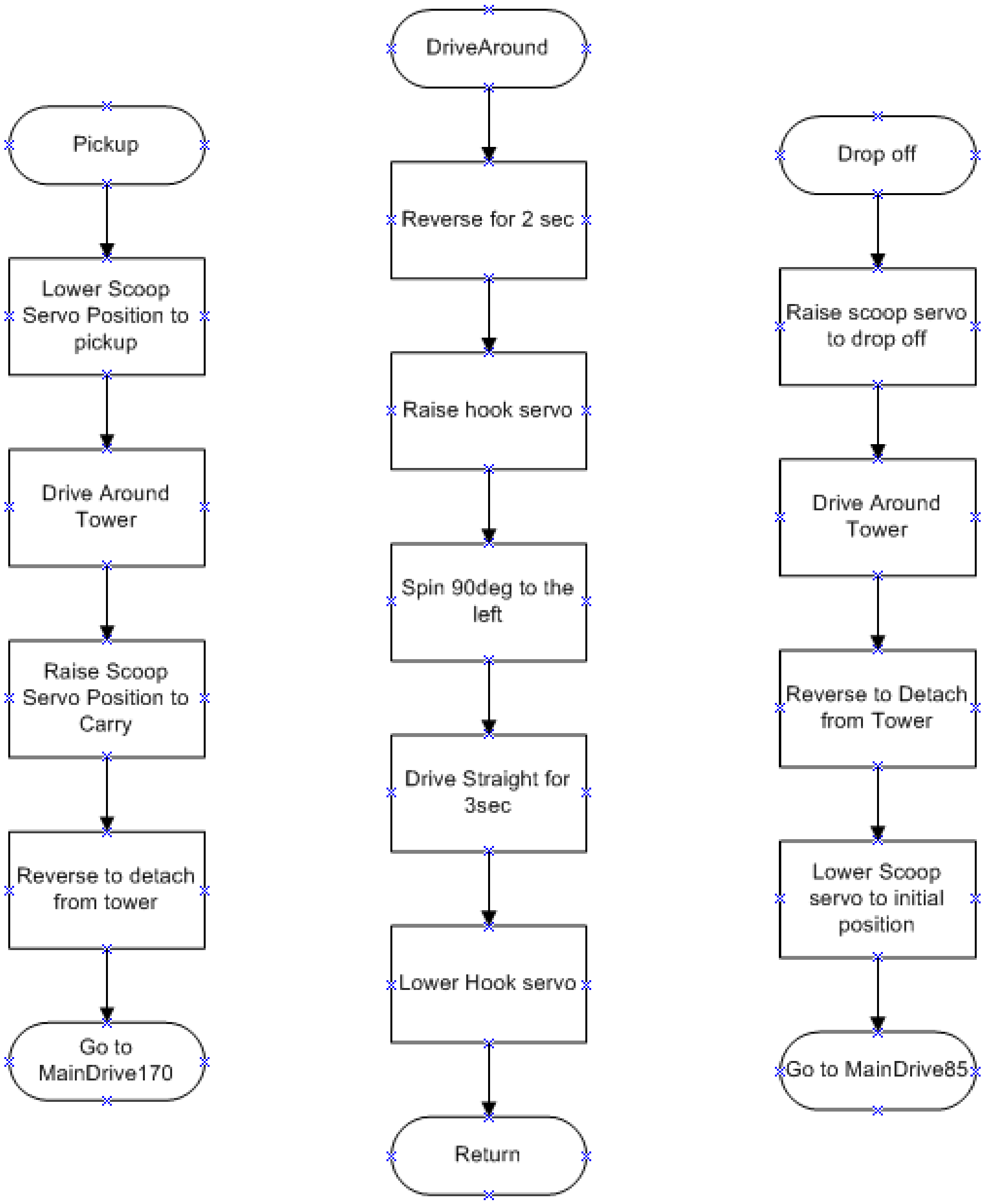
MME 4487a WebCT site

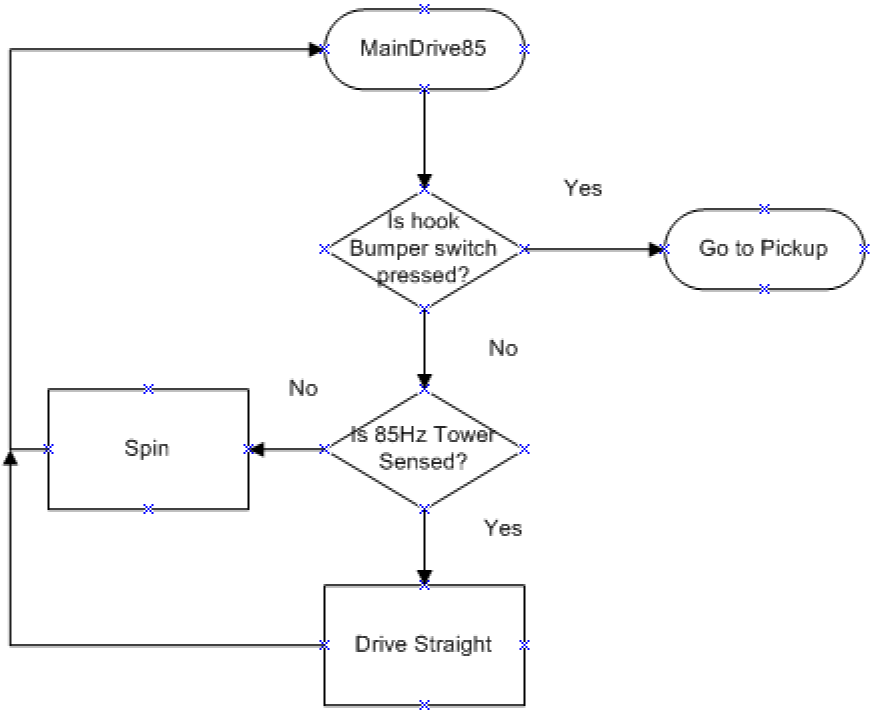
PIC16F87X Instruction Summary

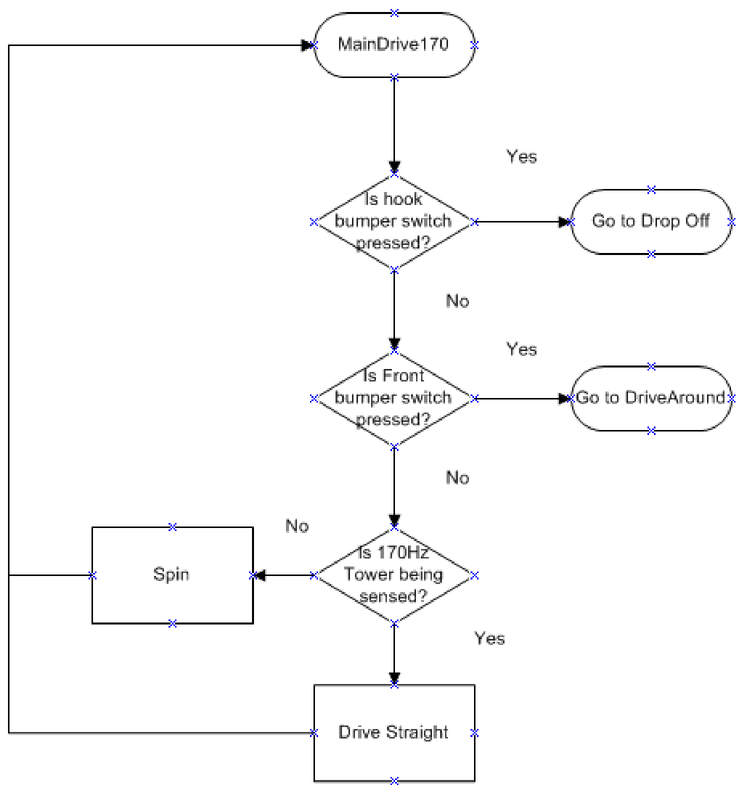
PIC16F87X Datasheet

Appendix A -1 Flowcharts




Pickup
Lower Scoop Servo Position to pickup
Drive Around Tower
Raise Scoop Servo Position to Carry
Reverse to detach from tower
Go to MainDrive170
DriveAround
Reverse for 2 sec
Raise hook servo
Spin 90deg to the left
Drive Straight for 3sec
Lower Hook servo
Return
Drop off
Raise scoop servo to drop off
Drive Around Tower
Reverse to Detach from Tower
Lower Scoop servo to initial position
Go to MainDrive85


MainDrive85
Yes
Is hook Bumper switch pressed?
Go to Pickup
No
No
Spin
Is 85Hz Tower Sensed?
Yes
Drive Straight


MainDrive170
Yes
Is hook bumper switch pressed?
Go to Drop Off
No
Yes
Is Front bumper switch pressed?
Go to DriveAround
No
No
Spin
Is 170Hz Tower being sensed?
Yes
Drive Straight

Appendix B – Program Code

```
list P="16F877A"
include <P16f877.inc>
__CONFIG _RC_OSC & _WDT_OFF & _PWRTE_ON & _BODEN_OFF & _LVP_OFF & _WRT_ENABLE_ON &
_CPD_OFF & _CP_OFF & _DEBUG_ON
;
;------------------------------------------------------------------------------
;       user variables
                        UDATA
Count           res 1           ; How long to drive straight forward
Count2          res 1
Count3          res 1
Pwm1            res 1   ; PWM from AN0 to send out CCP1 pin
Pwm2            res 1   ; PWM from AN1 to send out CCP2 pin
Tower           res 1           ;Defines which tower we are currently at
;beacon detect variables
Tmr1_ValL       res 1           ; store value from Timer 1 Least Significant Byte (LSB)
Tmr1_ValH       res 1           ; store value from Timer 1 Most Significant Byte (MSB)
TMR1_Status res 1               ; store status of IServ (i.e start or end of period)
BeaconStatus res 1              ; status of beacons that are being seen by the TSOP1238
PCLATH_TEMP     res 1           ; On interrupt the PCLATH, STATUS, and WREG registers
STATUS_TEMP     res 1           ; may be modified, it is good practice to store and
W_TEMP                  res 1           ; load these variables at the beginning and end of an
                                        ; interrupt routine. They are easiest to use as
MACROS.
;servo variables
TimerLen        res 1           ; Value for setting pulse width for hook servo
TimerLen1       res 1           ; Scoop servo
TimerLen2       res 1           ; Sensor Servo
Tmr0_Val        res 1           ; Actual value to write to timer 0
TmrState        res 1           ; determines state(ON/OFF) for RC Servo Signal
;------------------------------------------------------------------------------
; Locates startup code @ the reset vector
STARTUP         CODE
        nop                                     ; Reset vector
        goto    Start
        nop
        nop
        goto    IServ                   ; Interrupt vector
PROG1   CODE                    ; Locates main code
;------------------------------------------------------------------------------
;       start up and main routine code
Start
        nop                                     ; needed to avoid problems with ICD
        nop
        call    Init                    ; configure microcontroller hardware
                                        ; and initializing variable presets
Main
        call    HookDeploy
        call    MainDrive85
        goto    Main                    ; repeat
```

```
;-------------------------------------------------------------------------------
Init
        bcf             INTCON,7        ; Disable all interrupts while initializing
        call    Init_IO         ; Configure ADC(PORTA), digital outputs(PORTC)
        call    Init_PWM        ; Configure PWM
        bcf             PORTC,0         ; Set direction of right(Pwm2) dc motor (RC0)
        bcf             PORTC,3         ; Set direction of left(Pwm1) dc motor (RC3)
        call    Init_Timer1             ; configure and initialize Timer 1
        call    Init_Timer0             ; configure Timer0 to interrupt on overflow
        call    Init_Interrupts ; configure interrupts
        clrf    TmrState                ; Clear timer state to initialze
        return
;-------------------------------------------------------------------------------
Init_IO
        BANKSEL         TRISA           ; switch to bank 1 of memory map
        movlw   b'11111111'     ; configure PORTA as input channels
        movwf   TRISA           ; which will be used for analog inputs
        movlw   B'00000000'     ; setup 8 A/D channels with voltage
        movwf   ADCON1          ; reference of Vss and Vdd (0-5 volts)
        clrf    TRISC           ; configure PORTC as digital outputs
        BANKSEL         PORTC           ; switch back to bank 0 of memory map
        clrf    PORTC           ; turn off all PORTC outputs
;beacon check IO and Servo IO
        BANKSEL         TRISB                   ; switch to bank 1 of memory map
        movlw   b'11101001'             ; configure PORTB as input channels from which
        movwf   TRISB                   ; RB0/INT will be used for the TSOP1238 signal and RB1, RB2
and RB4 for servo's
        movlw   b'00011000'             ; configure PORTD as output channels which will
        movwf   TRISD                   ; display the results of the Beacons being sensed
        BANKSEL         PORTD                   ; on PORTD LEDs and initilize all LEDs to be off
        clrf    PORTD                   ;PortD, 3 was made an input for the bumper switch (hook)
        clrf    PORTB                   ;PortD, 4 was made an input for another bumper switch
(front bumper)
        return
;-------------------------------------------------------------------------------
Init_PWM
        BANKSEL         PIE1            ; switch to bank 1 of memory map
        bcf             PIE1,1          ; dis   able timer 2 interrupts
        bcf             PIE1,2          ; disable CCP1 interrupts
        bcf             PIE2,0          ; disable CCP2 interrupts
        BANKSEL         CCP1CON         ; switch back to bank 0 of memory map
        clrf    CCP1CON                 ; CCP1 module off (resets CCP1)
        clrf    CCP2CON                 ; CCP2 module off (resets CCP2)
        BANKSEL         PR2                     ; switch back to bank 1 of memory map
        movlw   0xff            ; decimal 255
        movwf   PR2             ; load period register
        BANKSEL         CCPR1L          ; switch back to bank 0 of memory map
        clrf    CCPR1L  ; clear msb's of duty cycle register for PWM1
        clrf    CCPR2L  ; clear msb's of duty cycle register for PWM2
        movlw   b'00000001' ; prescaler = 1:4,
        movwf   T2CON   ; turn off timer 2
        clrf    TMR2    ; clear/reset timer 2
        movlw   b'00001100' ; set CCP1 to PWM mode and enable
```

```
	movwf	CCP1CON	; or turn module on
	movlw	b'00001100' ; set CCP2 to PWM mode and enable
	movwf	CCP2CON		; or turn module on
	bsf		T2CON,2	; turn on timer 2
	return
;-----------------------------------------------------------------------------
Init_Timer1
	BANKSEL		T1CON				; switch to bank 0 of memory map
	movlw	b'00000000'			; setup timer 1 with 1:1 prescaler
	movwf	T1CON				; and disable/stop timer 1
	BANKSEL		PIE1		; switch to bank 1 of memory map
	bsf		PIE1,0				; enable Timer 1 Overflow Interrupt
	BANKSEL		PIR1				; switch to bank 0 of memory map
	bcf		PIR1,0				; clear Timer 1 Overflow Interrupt Flag bit
	clrf	TMR1L				; clear/initialize timer 1 LSB holding register
	clrf	TMR1H				; clear/initialize timer 1 MSB holding register
	clrf	Tmr1_ValL			; clear/initialize user Tmr1_ValL register
   clrf	Tmr1_ValH			; clear/initialize user Tmr1_ValH register
	clrf	TMR1_Status			; clear/initialize user TMR1_Status register
	clrf	BeaconStatus	; clear/initialize user BeaconStatus register
	return
;-----------------------------------------------------------------------------
Init_Timer0
	BANKSEL		OPTION_REG			; switch to bank 1 of memory map
	movlw	b'00000111'			; Setup internal clock, full prescaler (1:256)
	movwf	OPTION_REG
	BANKSEL TMR0				; switch back to bank 0 of memory map
	clrf	TMR0				; Clear timer
	return
;-----------------------------------------------------------------------------
Init_Interrupts
	BANKSEL		OPTION_REG		; switch to bank 1 of memory map
	bcf		OPTION_REG,6	; Setup RB0/INT to interrupt on falling edge
	BANKSEL INTCON				; switch to bank 0 of memory map
	movlw	b'11110000'			; enable RBO/INT external interrupt and enable
	movwf	INTCON				; Peripheral and Global unmasked interrupts as well as Timer0
interrupt
	return
;-----------------------------------------------------------------------------
HookDeploy
	movlw	d'160'				;deploy hook to extended
	movwf	TimerLen
;	movlw	d'235'				;deploy scoop to beginning position
	movlw	d'235'
	movwf	TimerLen1
	movlw	d'130'				; deploy sensor at beginning of the program
	movwf	TimerLen2
	return
;-----------------------------------------------------------------------------
Pickup_Lower
	movlw	d'210'				;move scoop to pickup position while hooking around tower
	movwf	TimerLen1
	return
```

```
;-----------------------------------------------------------------------------
Pickup_Carry
        ;movlw  d'165'                  ;move scoop to carry position whith rocks and moving to drop
off location
        movlw   d'160'
        movwf   TimerLen1
        return
;-----------------------------------------------------------------------------
Scoop_Drop
        movlw   d'120'                  ;move scoop to drop off position
        movwf   TimerLen1
        return
;-----------------------------------------------------------------------------
Hook_Middle
        movlw   d'190'                  ;deploy hook to extended
        movwf   TimerLen
        return
;-----------------------------------------------------------------------------
Hook_Down
        movlw   d'160'                  ; hook to extended
        movwf   TimerLen
        return
;-----------------------------------------------------------------------------

MainDrive85
        movlw   d'85'
        movwf   Tower
        btfss   PORTD,3         ;check if bumper has been hit, skip if not hit
        goto    TowerCheck              ;dummy sub routine for bumper testing
        call    BeaconCheck
        movfw   BeaconStatus
        movwf   PORTD
        movf    Tmr1_ValH,W     ; display the status of whether a Beacon is being
        sublw   d'37'                   ; checks to see if result zero then 85 Hz detect
        btfsc   STATUS,2                ; check if condition has been satisfied
        call    DriveStraight                   ; if condition met then drive straight
        call    Spin
        goto    MainDrive85
;-----------------------------------------------------------------------------
MainDrive170
        movlw   d'170'
        movwf   Tower
        btfss   PORTD,3                 ;check if bumper has been hit, skip if not hit
        goto    TowerCheck              ;sends to towercheck subroutine
        btfss   PORTD,4                 ;check if bumper has been hit, skip if not hit
        call    Around_Tower    ;sends to drive around tower subroutine
        call    BeaconCheck
        movfw   BeaconStatus
        movwf   PORTD
        movf    Tmr1_ValH,W             ; display the status of whether a Beacon is being
        sublw   d'18'                   ; checks to see if result zero then 85 Hz detect
        btfsc   STATUS,2                ; check if condition has been satisfied
        call    DriveStraight
```

```
        call    Spin
        goto    MainDrive170

;-----------------------------------------------------------------------------
;       update Pwm1 and Pwm2 so hardware produces PWM output signals
DriveStraight
        bcf             PORTC,0         ; Set direction of right(Pwm2) dc motor (RC0) reverse
        bcf             PORTC,3         ; Set direction of left(Pwm1) dc motor (RC3) forward
        movlw   d'165'          ;driving to the left slightly so that if we lose the tower we do not spin
360deg
        movwf   Pwm1            ;right side
        movlw   d'185'
        movwf   Pwm2            ;left side
        movf    Pwm1,W                  ; move PWM value into the register
        movwf   CCPR1L          ; which controls PWM hardware module
        movf    Pwm2,W                  ; move PWM value into the register
        movwf   CCPR2L          ; which controls PWM hardware module
 ;      movlw   b'00001100'     ;amount of time to delay approx 3sec
        movlw   b'00000111'
        movwf   Count3
        call    DriveDelay
        return                          ;returns to MainDrive
;-----------------------------------------------------------------------------
Drive_Pickup
        bcf             PORTC,0         ; Set direction of right(Pwm2) dc motor (RC0) reverse
        bcf             PORTC,3         ; Set direction of left(Pwm1) dc motor (RC3) forward
        movlw   d'185'
        movwf   Pwm1            ;right side
        movwf   Pwm2            ;left side
        movf    Pwm1,W                  ; move PWM value into the register
        movwf   CCPR1L          ; which controls PWM hardware module
        movf    Pwm2,W                  ; move PWM value into the register
        movwf   CCPR2L          ; which controls PWM hardware module
        movlw   b'00111111'     ;amount of time to delay approx 5sec
        movwf   Count3
        call    DriveDelay
        return                          ;returns to MainDrive
;-----------------------------------------------------------------------------
Drive_Drop_Off
        bcf             PORTC,0         ; Set direction of right(Pwm2) dc motor (RC0) reverse
        bcf             PORTC,3         ; Set direction of left(Pwm1) dc motor (RC3) forward
        movlw   d'185'
        movwf   Pwm1            ;right side
        movwf   Pwm2            ;left side
        movf    Pwm1,W                  ; move PWM value into the register
        movwf   CCPR1L          ; which controls PWM hardware module
        movf    Pwm2,W                  ; move PWM value into the register
        movwf   CCPR2L          ; which controls PWM hardware module
        movlw   b'00111111'     ;amount of time to delay approx 10sec
        movwf   Count3
        call    DriveDelay
        return                          ;returns to MainDrive
;-----------------------------------------------------------------------------
```

```
Spin
	bcf		PORTC,0		; Set direction of right(Pwm2) dc motor (RC0) reverse
	bsf		PORTC,3		; Set direction of left(Pwm1) dc motor (RC3) forward
	movlw	d'200'
	movwf	Pwm1		;right side
	movwf	Pwm2		;left side
	movf	Pwm1,W			; move PWM value into the register
	movwf	CCPR1L		; which controls PWM hardware module
	movf	Pwm2,W			; move PWM value into the register
	movwf	CCPR2L		; which controls PWM hardware module
	return				;return to MainDrive85 to detect towers
;-----------------------------------------------------------------------------
Pick_Up
	call Pickup_Lower		;lowers scoop to pickup angle
	call Drive_Pickup ;Drives around tower at choosen speed and amount of time
	call Pickup_Carry ;raises scoop to carry angle
	call Reverse			;used to detatch hook from tower
;	call Hook_Middle ;moves hook above middle of robot
	goto MainDrive170		;to drive to drop off point

;-----------------------------------------------------------------------------
Drop_Off
	call Scoop_Drop			;lowers scoop to pickup angle
	call Drive_Drop_Off		;Drives around tower at choosen speed and amount of time
	call Reverse
	call HookDeploy
	goto MainDrive85

;-----------------------------------------------------------------------------
Reverse
	bsf		PORTC,0		; Set direction of right(Pwm2) dc motor (RC0) reverse
	bsf		PORTC,3		; Set direction of left(Pwm1) dc motor (RC3) forward
	movlw	d'170'
	movwf	Pwm1		;right side
	movlw	d'200'
	movwf	Pwm2		;left side
	movf	Pwm1,W			; move PWM value into the register
	movwf	CCPR1L		; which controls PWM hardware module
	movf	Pwm2,W			; move PWM value into the register
	movwf	CCPR2L		; which controls PWM hardware module
	movlw	b'00001111'	; amount of time to delay approx 3sec
	movwf	Count3
	call	DriveDelay
	return
;-----------------------------------------------------------------------------
Around_Tower
	call Reverse_Around				;reverse for 3 sec
	call Hook_Middle			; hook up straight
	call Spin_90					;spins to the right 90deg
	call DriveStraight_Around ;drives straight for 3 sec, same routine used in main drive subroutines
	call Hook_Down					;deploy hook down
	return
```

```
;-------------------------------------------------------------------------------
Spin_90
        bsf         PORTC,0         ; Set direction of right(Pwm2) dc motor (RC0) forward
        bcf         PORTC,3         ; Set direction of left(Pwm1) dc motor (RC3)  reverse
        movlw   d'200'
        movwf   Pwm1            ;right side
        movwf   Pwm2            ;left side
        movf    Pwm1,W                  ; move PWM value into the register
        movwf   CCPR1L          ; which controls PWM hardware module
        movf    Pwm2,W                  ; move PWM value into the register
        movwf   CCPR2L          ; which controls PWM hardware module
        movlw   b'00010100'     ; amount of time to delay approx 5sec
        movwf   Count3
        call    DriveDelay
        return                          ;return to MainDrive85 to detect towers
;-------------------------------------------------------------------------------
Reverse_Around
        bsf         PORTC,0         ; Set direction of right(Pwm2) dc motor (RC0) reverse
        bsf         PORTC,3         ; Set direction of left(Pwm1) dc motor (RC3) forward
        movlw   d'170'
        movwf   Pwm1            ;right side
        movlw   d'200'
        movwf   Pwm2            ;left side
        movf    Pwm1,W                  ; move PWM value into the register
        movwf   CCPR1L          ; which controls PWM hardware module
        movf    Pwm2,W                  ; move PWM value into the register
        movwf   CCPR2L          ; which controls PWM hardware module
        movlw   b'00001000'     ; amount of time to delay approx 3sec
        movwf   Count3
        call    DriveDelay
        return
;-------------------------------------------------------------------------------
DriveStraight_Around
        bcf         PORTC,0         ; Set direction of right(Pwm2) dc motor (RC0) reverse
        bcf         PORTC,3         ; Set direction of left(Pwm1) dc motor (RC3) forward
        movlw   d'185'          ;driving to the right slightly
        movwf   Pwm1            ;right side
        movlw   d'175'
        movwf   Pwm2            ;left side
        movf    Pwm1,W                  ; move PWM value into the register
        movwf   CCPR1L          ; which controls PWM hardware module
        movf    Pwm2,W                  ; move PWM value into the register
        movwf   CCPR2L          ; which controls PWM hardware module
        movlw   b'00011100'     ;amount of time to delay approx 3sec
        movwf   Count3
        call    DriveDelay
        return                          ;returns to MainDrive
;-------------------------------------------------------------------------------
DriveDelay

        movlw   b'11111111'     ; initialize the variable Count to
        movwf   Count           ; start at the value of 128
        movlw   b'11111111'
```

```
        movwf   Count2

Loop3
Loop2
Loop
        decfsz  Count,F         ; exit the loop if Count is equal to
        goto    Loop            ; zero and display that on PORTD
        movlw   b'11111111'
        movwf   Count
        decfsz  Count2,F
        goto    Loop2
        movlw   b'11111111'
        movwf   Count2
        decfsz  Count3,F
        goto    Loop3
        return

;-------------------------------------------------------------------------------
TowerCheck                              ;Determines the current tower
        movfw   Tower
        sublw   d'85'
        btfsc   STATUS,2
        goto    Pick_Up
        movfw   Tower
        sublw   d'170'
        btfsc   STATUS,2
        goto    Drop_Off
        return

;-------------------------------------------------------------------------------

BeaconCheck                                     ; needs to be modified in order to determine which
                                                        ; beacon signal is being received if any
T_85Hz
        nop
        movf    Tmr1_ValH,W             ; Move high value of timer into W. Reg.
        sublw   d'37'                   ; Subtract Tmr1_ValH against literal value determined in lab
(37)
        btfsc   STATUS,2                ; Check the zero bit to determine whether signal is 85 kHz
        bsf             BeaconStatus,1  ;
        goto    T_170Hz                 ; Beacon was not 85 Hz so test for 170 Hz
        return                                  ; and update BeaconStatus if necessary
T_170Hz
        nop
        movf    Tmr1_ValH,W             ; Move high value of timer into W. Reg.
        sublw   d'18'                   ; Subtract Tmr1_ValH against literal value determined in lab
(18)
        btfsc   STATUS,2                ; Check the zero bit to determine whether signal is 85 kHz
        bsf             BeaconStatus,2  ;
        goto    T_340Hz                                         ; need to modify to check for
170Hz signal
        return                                  ; and update BeaconStatus if necessary
T_340Hz
```

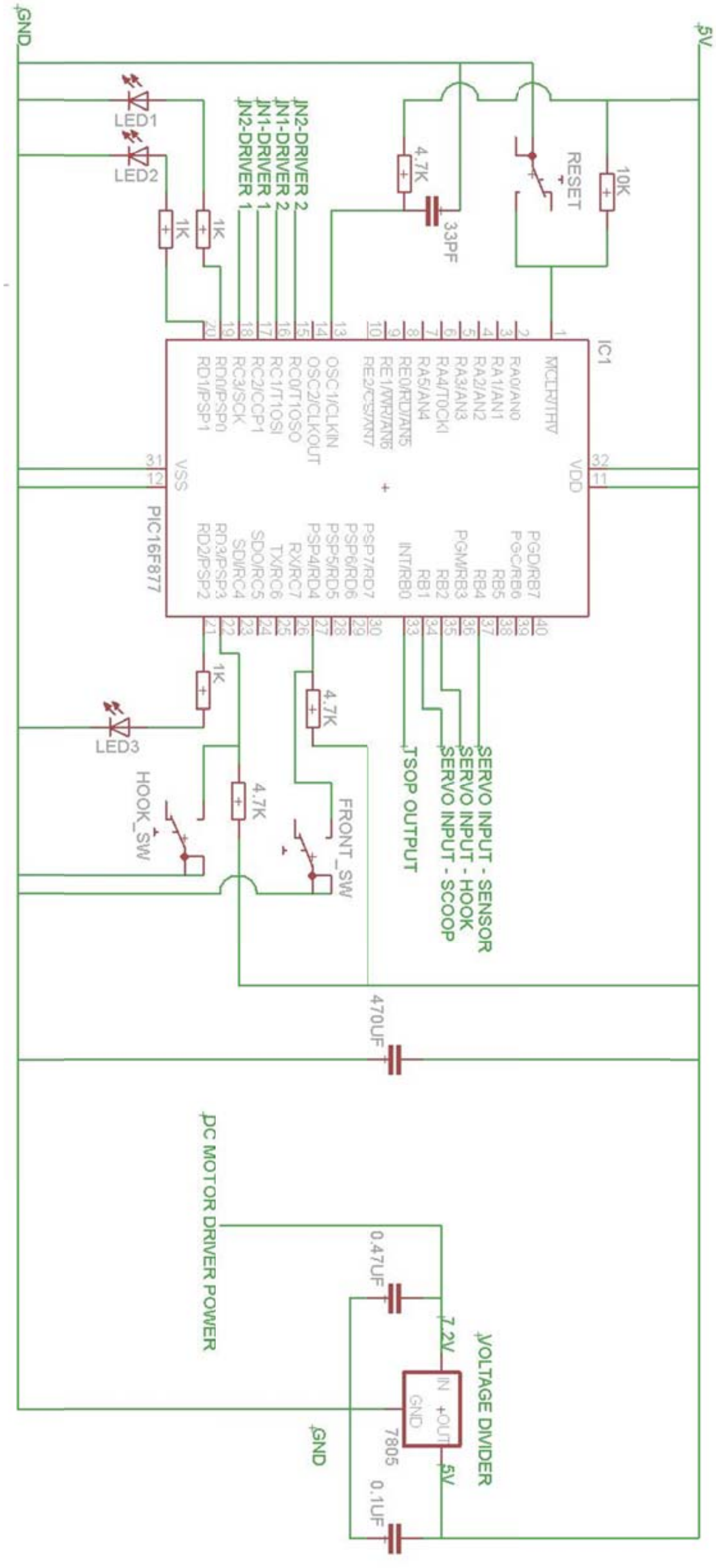
```
        nop
        movf    Tmr1_ValH,W             ; Move high value of timer into W. Reg.
        sublw   d'9'                    ; Subtract Tmr1_ValH against literal value determined in lab
(9)
        btfsc   STATUS,2                ; Check the zero bit to determine whether signal is 85 kHz
        bsf             BeaconStatus,3  ;
        goto    NoBeacon                                        ; need to modify to check
for 340Hz signal
        return                                  ; and update BeaconStatus if necessary
NoBeacon
        nop
        return                                  ; and update BeaconStatus if necessary
;---------------------------------------------------------------------------
PUSH_MACRO      MACRO
        movwf   W_TEMP                          ; copy W to W_TEMP register
        swapf   STATUS,W                ; swap status to be saved into W
        movwf   STATUS_TEMP             ; save status to bank 0 STATUS_TEMP register
        movf    PCLATH,W
        movwf   PCLATH_TEMP             ; save PCLATH from W
        ENDM
;---------------------------------------------------------------------------
POP_MACRO       MACRO
        movf    PCLATH_TEMP,W   ; restore PCLATH
        movwf   PCLATH                  ; move W into PCLATH
        swapf   STATUS_TEMP,W   ; swap STATUS_TEMP register into W
        movwf   STATUS                  ; move W into STATUS register
        swapf   W_TEMP,F                ; swap W_TEMP
        swapf   W_TEMP,W                ; swap W_TEMP into W
        ENDM
;---------------------------------------------------------------------------
IServ
        PUSH_MACRO                              ; MACRO saves context registers, or in-line code
        BANKSEL INTCON                  ; switch to bank 0 of memory map
        btfss   INTCON,1                ; check if RB0/INT Interrupt Flag Bit is set
        goto    TMR1_INTF               ; NO; check for timer 1 overflow interrupt flag
        bcf             INTCON,4                ; YES; then disable RB0/INT external interrupt
        btfsc   TMR1_Status,0   ; determine if timers should be started or stopped
        goto    StopTimer       ; timer is started; so stop timer; end of period
StartTimer                                      ; start of period for the signal
        clrf    TMR1L                   ; initialize/clear timer 1 LSB holding register
        clrf    TMR1H                   ; initialize/clear timer 1 MSB holding register
        bsf             T1CON,0                 ; start/enable Timer 1 to capture the period

        bsf     TMR1_Status,0   ; indicate that timer 1 is started so next time an
        goto    end_RBOIServ    ; RBO/INT interrupt occurs its the end of period
StopTimer
        BANKSEL         TMR1L                   ; switch to bank 0 of memory map
        movf    TMR1L,W                 ; store a copy of the timer 1 results; full period
        movwf   Tmr1_ValL               ; store a copy of TMR1 low-byte holding register
        movf    TMR1H,W                         ; store a copy of the timer 1 results; full period
        movwf   Tmr1_ValH               ; store a copy of TMR1 high-byte holding register
        clrf    TMR1L                   ; clear/initialize timer 1 LSB holding register
        clrf    TMR1H                   ; clear/initialize timer 1 MSB holding register
```

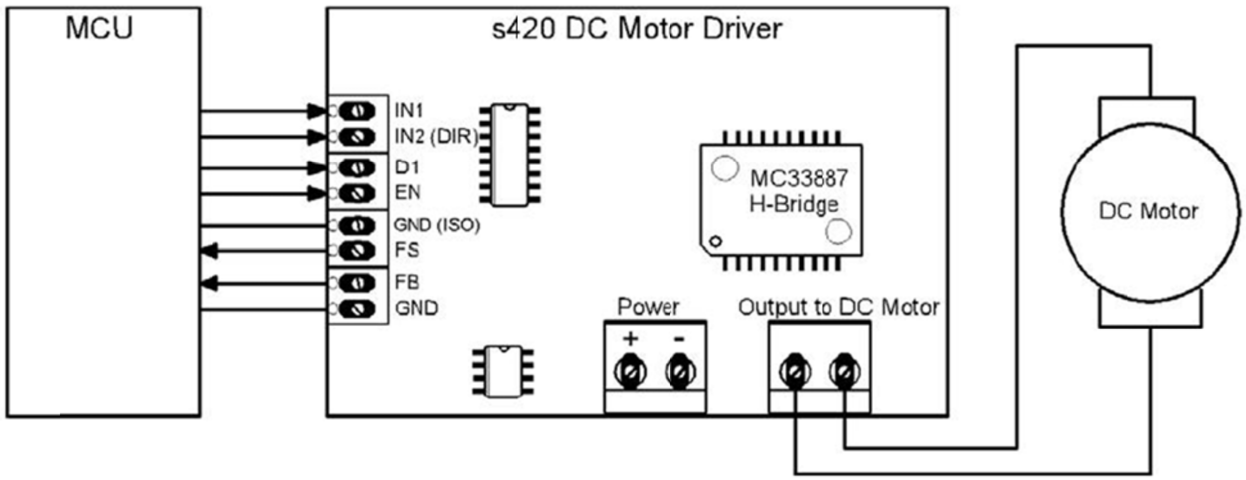
```
	bcf	TMR1_Status,0	; initialize status of IServ for next signal seen
end_RBOIServ
	bcf		INTCON,1		; reset RBO/INT Interrupt Flag Bit
	bsf		INTCON,4		; enable RB0/INT Interrupt
	goto	end_IServ
TMR1_INTF
	BANKSEL		PIR1	; switch to bank 0 of memory map
	btfss	PIR1,0			; check for Timer 1 Overflow Interrupt Flag
	goto	Check_TMR0	; go to Check_TMR0 if Timer 1 did not overflow
	bcf		T1CON,0		; yes; stop/disable Timer 1
	bcf		PIR1,0		; clear TMR1 Overflow Interrupt Flag Bit
	clrf	BeaconStatus	; clear BeaconStatus to indicate no signals found
	clrf	Tmr1_ValL		; initialize timer 1 holding registers
	clrf	Tmr1_ValH		; for the next time a signal is seen
	goto	end_IServ
;Timer0 interrupt for servo's
Check_TMR0
	BANKSEL INTCON			; may not need this
	btfss	INTCON,2		; Check whether the TMR0 Interrupt Flag Bit is set
	goto	end_IServ		; NO; then end IServ; we only care about TMR0 INT.
	bcf		INTCON,5		; YES; then disable TMR0 interrupt
	movlw	0x00			; decides current timer state that is in process
	subwf	TmrState,W		; in order to properly drive the RC Servo
	btfsc	STATUS,Z
	goto	Timer_Pot		; this will set RC Servo signal HIGH/ON
	movlw	0x01			; which sets the pulse width
	subwf	TmrState,W
	btfsc	STATUS,Z		; this will set the RC Servo signal LOW/OFF
	goto	Timer_Low		; which basically sets the period length
	movlw	0x02			; decides current timer state that is in process
	subwf	TmrState,W		; in order to properly drive the RC Servo
	btfsc	STATUS,Z
	goto	Timer_Pot1		; this will set RC Servo signal HIGH/ON
	movlw	0x03			; which sets the pulse width
	subwf	TmrState,W
	btfsc	STATUS,Z		; this will set the RC Servo signal LOW/OFF
	goto	Timer_Low1		; which basically sets the period length
	movlw	0x04			; decides current timer state that is in process
	subwf	TmrState,W		; in order to properly drive the RC Servo
	btfsc	STATUS,Z
	goto	Timer_Pot2		; this will set RC Servo signal HIGH/ON
	movlw	0x05			; which sets the pulse width
	subwf	TmrState,W
	btfsc	STATUS,Z		; this will set the RC Servo signal LOW/OFF
	goto	Timer_Low2		; which basically sets the period length
Timer_Pot
	bsf		PORTB,1			; set signal HIGH/ON to RC servo
	BANKSEL		OPTION_REG		; switch to bank 1 of memory map
	bsf		OPTION_REG,1	; make sure prescaler is 1:16
	bcf		OPTION_REG,2
	BANKSEL		PORTA			; switch back to bank 0 of memory map
	movfw	TimerLen		; recall value from ANO
	movwf	Tmr0_Val		; used to set time for signal to be ON (pulse width)
```

```
        incf    TmrState                ; set up for next TMR0 Interrupt State
        goto    Start_Timer             ; will setup TMR0 for next Interrupt
Timer_Low
        bcf             PORTB,1                 ; set signal LOW/OFF to RC servo
        clrf    Tmr0_Val                ; set time for signal to be OFF
        BANKSEL         OPTION_REG              ; switch to bank 1 of memory map
        bcf             OPTION_REG,1    ; Change prescaler to 1:64
        bsf             OPTION_REG,2    ; for long off time (basically sets period)
        BANKSEL         PORTA                   ; switch back to bank 0 of memory map
        incf    TmrState                ; set up for next TMR0 Interrupt State
        goto    Start_Timer             ; will setup TMR0 for next Interrupt
Timer_Pot1
        bsf             PORTB,2                 ; set signal HIGH/ON to RC servo
        BANKSEL         OPTION_REG              ; switch to bank 1 of memory map
        bsf             OPTION_REG,1    ; make sure prescaler is 1:16
        bcf             OPTION_REG,2
        BANKSEL         PORTA                   ; switch back to bank 0 of memory map
        movfw   TimerLen1               ; recall value from ANO
        movwf   Tmr0_Val                ; used to set time for signal to be ON (pulse width)
        incf    TmrState                ; set up for next TMR0 Interrupt State
        goto    Start_Timer             ; will setup TMR0 for next Interrupt
Timer_Low1
        bcf             PORTB,2                 ; set signal LOW/OFF to RC servo
        clrf    Tmr0_Val                ; set time for signal to be OFF
        BANKSEL         OPTION_REG              ; switch to bank 1 of memory map
        bcf             OPTION_REG,1    ; Change prescaler to 1:64
        bsf             OPTION_REG,2    ; for long off time (basically sets period)
        BANKSEL         PORTA                   ; switch back to bank 0 of memory map
        incf    TmrState                ; set up for next TMR0 Interrupt State
        goto    Start_Timer             ; will setup TMR0 for next Interrupt
Timer_Pot2
        bsf             PORTB,4                 ; set signal HIGH/ON to RC servo
        BANKSEL         OPTION_REG              ; switch to bank 1 of memory map
        bsf             OPTION_REG,1    ; make sure prescaler is 1:16
        bcf             OPTION_REG,2
        BANKSEL         PORTA                   ; switch back to bank 0 of memory map
        movfw   TimerLen2               ; recall value from ANO
        movwf   Tmr0_Val                ; used to set time for signal to be ON (pulse width)
        incf    TmrState                ; set up for next TMR0 Interrupt State
        goto    Start_Timer             ; will setup TMR0 for next Interrupt
Timer_Low2
        bcf             PORTB,4                 ; set signal LOW/OFF to RC servo
        clrf    Tmr0_Val                ; set time for signal to be OFF
        BANKSEL         OPTION_REG              ; switch to bank 1 of memory map
        bcf             OPTION_REG,1    ; Change prescaler to 1:64
        bsf             OPTION_REG,2    ; for long off time (basically sets period)
        BANKSEL         PORTA                   ; switch back to bank 0 of memory map
        clrf    TmrState                ; set up for next TMR0 Interrupt State
        goto    Start_Timer             ; will setup TMR0 for next Interrupt
Start_Timer
        movfw   Tmr0_Val                ; recall current time value to be set
        movwf   TMR0                    ; setup up timer start value
        bcf             INTCON,2                ; reset TMR0 Overflow Interrupt Flag Bit
```

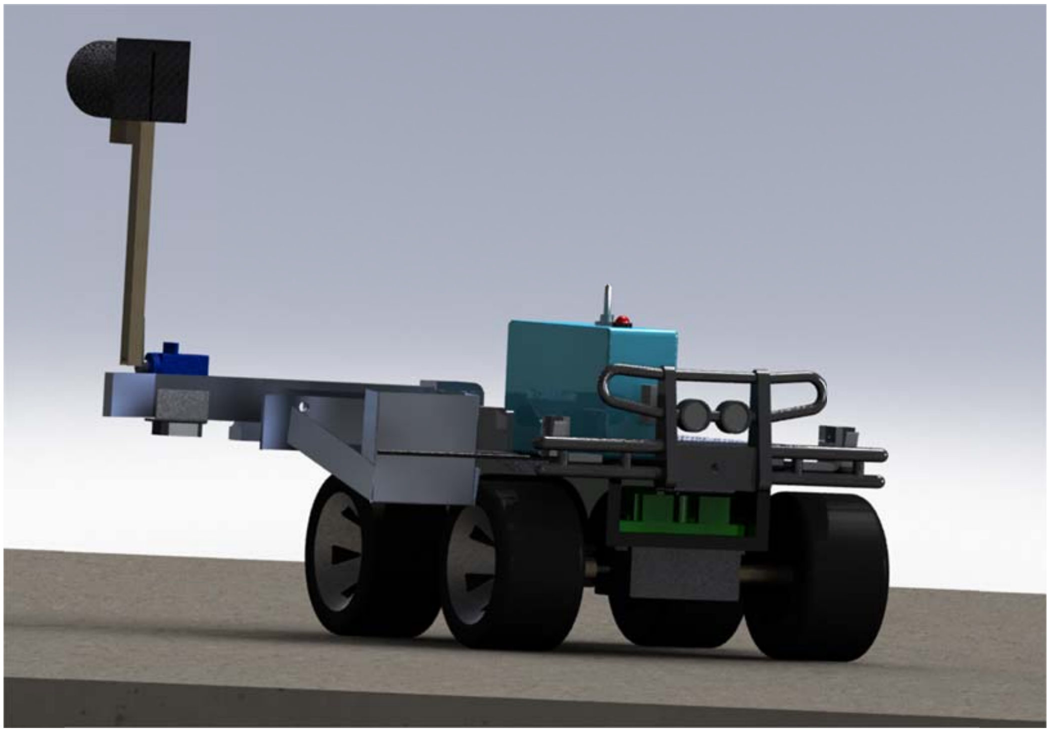
```
	bsf		INTCON,5		; enable TMR0 Interrupt
end_IServ
	POP_MACRO				; MACRO restores context registers, or in-line code
	retfie
	end
```

Appendix C – Circuit Diagram


GND
5V
LED1
LED2
LED3
1K
IN2-DRIVER 2
IN1-DRIVER 2
IN1-DRIVER 1
IN2-DRIVER 1
4.7K
33PF
RESET
10K
IC1
PIC16F877
HOOK_SW
FRONT_SW
TSOP OUTPUT
SERVO INPUT - SENSOR
SERVO INPUT - HOOK
SERVO INPUT - SCOOP
470UF
DC MOTOR DRIVER POWER
0.47UF
7.2V
VOLTAGE DIVIDER
7805
0.1UF


MCU
s420 DC Motor Driver
IN1
IN2 (DIR)
D1
EN
GND (ISO)
FS
FB
GND
MC33887
H-Bridge
Power
+
-
Output to DC Motor
DC Motor

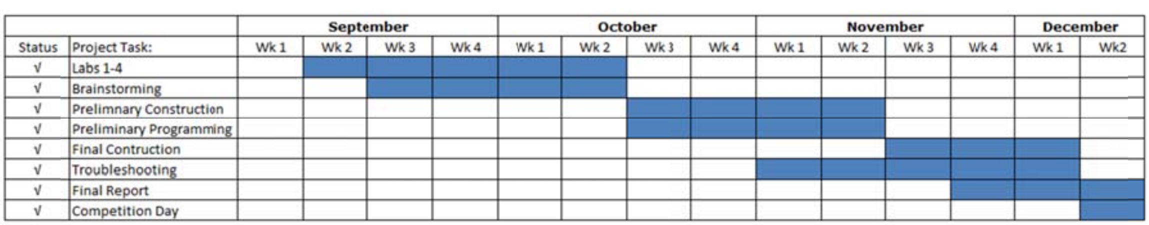
Appendix D – Detailed CAD Model



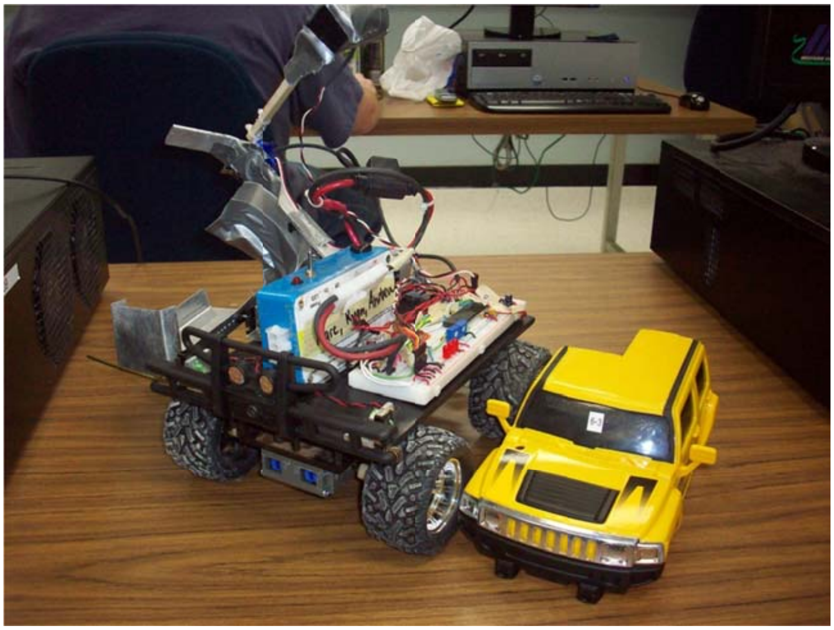
Appendix E - BOM

Component	Quantity	Unit Cost	Total cost
Frame	1	1.75	$1.75
Scoop	1	1.5	$1.50
Hook	1	2	$2.00
DC Motors	2	17	$34.00
RC Servos	3	22.79	$68.37
Nuts and Bolts	20	0.02	$0.40
Wire	10 Meters	0.0002/m	$0.00
Wheels	4	4	$16.00
10k resistor	1	0.26	$0.26
4.7k resitor	3	0.26	$0.78
1k resistor	3	0.26	$0.78
LED	3	0.18	$0.54
Switch	3	0.5	$1.50
33pF Capacitor	1	0.26	$0.26
470uF Capacitor	1	0.5	$0.50
0.47uF Capacitor	1	0.4	$0.40
0.1uF Capacitor	1	0.2	$0.20
Voltage Divider 7805	1	1.38	$1.38
Male Pin Connector	4	0.03	$0.12
PIC16F877A Microcontroller	1	2.55	$2.55
Sub Total			$133.29
Tax (13%)			$17.33
Total			$150.62

Appendix F – Gantt Chart (Project Shedule)



Appendix G – Picture of Robot



Appendix H

Drawings