Engineering Senior Design Reports

Engineering Science Department

4-28-2009

Mechatronic Camera Operator: Final Design Report

A. Hernandez
Trinity University

T. Nunamaker
Trinity University

J. Simpkins
Trinity University

W. Straw
Trinity University

A. Vaselaar
Trinity University

Repository Citation

Hernandez, A.; Nunamaker, T.; Simpkins, J.; Straw, W.; and Vaselaar, A., "Mechatronic Camera Operator: Final Design Report" (2009). *Engineering Senior Design Reports*. 26.
https://digitalcommons.trinity.edu/engine_designreports/26

TRINITY UNIVERSITY

Final Design Report

ENGR-4381

4/28/2009

Mechatronic Camera Operator

A. Hernandez, T. Nunamaker, J. Simpkins, W. Straw, A. Vaselaar

Dr. Aminian, Advisor

Abstract:

A mechatronic system is designed, constructed, and tested to aid filmmakers in the movement and control of a video camera. The system design allows for 6-DOF camera movement (movement in all three spatial directions, pan, tilt, and roll). The system is controlled by a human operator, using an implementation of a gamepad controller, and the system is battery-powered; the theoretical range of the system is therefore limited only by the onboard battery power, and the operator's ability to keep within cord-length of the system as it moves.

A misallocation of time resources resulted in an incomplete physical design, but preliminary testing indicates that the design is sound, and that mechanical specifications are sufficiently robust for a working final system. Further time and resources would be used to complete physical construction and electronic implementation, and to implement a feedback system to allow for closed-loop actuator control and the function of repeatable motion.

Executive Summary

The system should be able to move a camera through the same paths of motion that a human camera-man can move it through. Project constraints include a $2700 budget, a vertical camera elevation range of 2-6 ft and compatability with the filming environment. Design criteria include ease of set up, performance in a flat, closed room, silent system operation, ease of model development and physical construction, ease of setup and usage, mobility precision, mobility flexibility, and speed. The final implementation of the design is battery-powered and includes human-input through a gamepad controller to control each moving part, but does not employ any robotic features that allow for autonomous operation. The system fails in this respect by lacking the feature of repeatability and the ability to remember specific paths.

Mechanically, the base of the system consists of a plywood-laden steel frame built for structural robustness and is intended to make the system bottom-heavy to prevent tipping. Attached to the frame are two high-powered DC motors, each with an 8-inch diameter wheel, and a number of castor wheels for support.

The frame is two-tiered. On the lower tier electronics are safely housed; on the upper tier a turntable is attached to the base plywood. On the turntable a vertically-oriented threaded rod is mounted with two support rods all of which are connected with flat plates on top and bottom for structural support. A platform nut with a fixed horizontal orientation on the threaded rod changes elevation when the rod is rotated. Attached to the platform nut is a camera-head; this assembly is drived by a motor and results in changes of elevation for the camera.

Electrically and electronically, the motors are controlled by PIC microcontrollers. The PICs employ pulse-width-modulation and serial communication to control the different motors, and must communicate with each other to properly distribute commands from the PS2 controller. Implementing communication between the PICs is not completed and thus the final system is incomplete. Independently, each of the electronic subsystems works properly, with the exception of the portion that actuates the motor that turns the turntable.

Because the system is incomplete, limited testing has been performed. Mechanically, each system component works properly and exhibits structural soundness. Electronically, no failures are observed, but some components are incomplete. The threaded-rod motor makes use of the same microcontroller code as the wheel motors, so theoretically only a small barrier remains to make it functional. Additionally, finalizing PIC functionality to enable communication between chips should only require enabling built-in functionality and proper configuration, so this is also only a small barrier.

If further time and resources were allocated, they would be used to complete physical construction and electronic implementation, and to implement a feedback system to allow for closed-loop actuator control and the function of repeatable motion. There is every indication at this point that the design as it stands would be effective if completed.

Table of Contents

Table of Figures

1 Introduction

The greatest art does not feel like art, but simply a natural extension of the human experience. A film is most powerful when it feels most vivid, and special effects and effective use of camera control are tools that have been used towards this end since the advent of film-making. The Robotic Camera Operator group is working in this subset of the film industry: precise camera control, and by application of that control, special effects.

Creative expression drives film-making however too often the creative expression of filmmakers is stifled by the limited feasibility of the camera shots that they can use. There are often time, cost, physical, and human limitations that make certain shots out of reach for a particular film. The ultimate goal of this project is to improve the tools available to the filmmaker in order to allow for improved expression of ideas, much like expanding the palate of a painter. Additionally, our final system should allow such shots to be made with a smaller investment of time, money and effort: the painter's palette is no more useful if it is larger but prohibitively expensive or inaccessible.

There are two main solutions available at this time, the first being the motion control camera. This system consists of a very large remotely operated vehicle that rides on a set of tracks, similar to those of a train. This system allows repeatability, making compositing effects possible, yet it is still extremely difficult to employ. The computer interface for the system is not intuitive, and requires a specialized operator. Additionally, the necessary setup of equipment is time consuming, and the sheer cost the equipment makes such a system impractical for many filmmakers.

Another system available is the Steadicam which is an apparatus that a camera operator wears. Using this system removes the unintentional nuances of handheld camera motion; essentially, it is a low-pass filter for motion, removing the shake of the operator's hand but leaving the intentional camera movement. This allows for camera movement limited only by the operator's agility, with no time wasted setting up track. Unfortunately, the system is still susceptible to human error, and the camera movements are not precisely repeatable (thereby removing the option of doing two-dimensional compositing).

1.1 Problem Description

The objective of the robotic camera operator is to facilitate creative freedom in filmmaking. This is done by providing a robotic system to move the video camera through a scene precisely, both in timing and position. The motion must be able to be repeated as many times as is necessary without error. This is currently not possible without using complicated setups that are not feasible at certain locations or with limited budgets. By allowing directors to sculpt the camera movement precisely, the system will allow them the ability to execute elaborate shots quickly and easily.

For this particular year of the project, a design is proposed to construct a system that will allow these functions with a lightweight (approximately three pound) consumer camcorder. For subsequent work the system is designed to be easily scalable so that a second-generation prototype could perform the same function with a much heavier digital or film camera.

1.2 Constraints

The design is bound to several constraints; it is important to identify and design within these constraints in order to design a satisfactory solution for the problem. The constraints also serve to filter out problematic solutions, and allow for the desired use of the system to be demonstrated. There are several factors to be considered in setting the constraints, such as the allotted budget and factors specific to the film industry.

1.2.1 Budget

One of the greatest and most influential constraints on the design is the budget that has been made available for the project.

A relatively small amount of money has been allotted for the project so low design costs are an important constraint. Moreover, low design costs will ensure low production costs, which is desirable if the system is ever to be sold or rented to low budget film makers. The total budget of $2,700 includes $1,200 from the Engineering Department and $1,500 in grant money from IEEE Computer Society.

1.2.2 Performance

The camera should be able to film from altitudes ranging from 2 to 6 feet above the ground, as this approximates the range of the handheld camera operator that the design would ideally replace. In the event that the robotic camera operator is able to reach altitudes higher or lower, the constraint has been met and exceeded. This would simply serve to increase the value of the design as a creative tool. In the same vein it is also important that it can move a small camera similar to one a human operator would be able to control.

1.2.3 Environment

In the film industry, much of the filming is done in enclosed sound stages, which is the most probable location that the design will be used in. Therefore, it is imperative that the system be able to function in a small sound stage with a flat and level floor. For the purposes of this design, the sound stage size is minimized to 25 ft by 25 ft as the group has access to such a space which will prove helpful during testing.

1.2.4 Setup

It is important for the system to be simple to set up and use. This will allow the design to be feasible to the greatest amount of film-makers particularly those with tight time and budgetary constraints, such as small independent film makers. Moreover a system that is simple to set up and easy to use will lower production cost of the film, as less time is spent setting up each shot.

1.2.5 Sound

Finally, the proposed system must be relatively silent, as to not disturb filming. In essence the system should be inaudible to a microphone recording actors' dialogue; ideally the system would be completely silent.

1.3 Design Criteria

In order to select the most appropriate solution to the problem it is necessary to set working criteria. Although not as rigid as the constraints, the working criteria serve as a great mechanism in choosing a final solution. A more in-depth discussion of each criterion can be found below.

1.3.1 Development Requirements

It must be recognized that this project is not taking place in the professional field. The scope of project is large and the design team is relatively small with a limited amount of resources. For the greatest outcome, it is desirable to design regarding the abilites of the design team.

1.3.1.1 Model Development

This constraint deals with how simple the mathematical model of the movement of the system is to develop, which translates to a certain resource cost. A more simple mathematical model is more simple to develop, debug and implement. It will also limit the potential for human error in the design.

1.3.1.2 Construction

Construction is simply the complexity of the design and the skill it will take to build it. The simpler a system is to physically build, the more likely the prototype will be successful. This will also aid in keeping costs down if ever marketed, and the cost of employee resources needed.

1.3.2 Overall Ease of Use

To allow for film makers to easily adopt this new tool, it should not discourage an operator who may not be technically inclined. Moreover, the system should be able to withstand the environment in which it will be used.

1.3.2.1 Setup

This is based on how quickly a shot can be set up, as well as how quickly the system can be installed at a particular location. A design is considered sufficient if it has a setup time less than the comparable setup time for a track-and-dolly motion control camera; however, faster setup times are even more desirable because they result in less cost to the production. This leaves us with a target set up time of less than 1 hour[1].

1.3.2.2 Usage

The system should be as easy to use as possibly so as to minimize the time it takes to train a new operator, and to encourage non-technically inclined filmmakers to use the system.

This is measured on the perceived complexity it will take to program a shot with the system as well as to operate the camera while filming.

1.3.2.3 Durability

If the system is to be used in film making, it will undoubtedly be subjected to an assortment of adverse conditions. Even if these conditions are avoided, it is assumed that the system is to be used for long periods of time. It is important that the system be able to withstand the stresses of continual use, otherwise, the system may break and require repair, costing both time and money.

1.3.3 Creative Freedom

The primary purpose of the design is to free a film maker to easily replicate camera movements or otherwise move a camera in a space. The fewer limitations imposed upon the abilities of the camera, the more versatile and thus desirable the design.

1.3.3.1 Precision

Precision is based on how accurately the system can follow a preprogrammed path repetitively. The more precise a design, the better it is for compositing work and other special effects processes.

1.3.3.2 Mobility

Mobility is based upon the overall need to be able to move the camera from one point to another as easily as possible. The camera should be able to move to between any two points within a 25 ft by 25 ft area, and between 2 to 6 ft in elevation. The greater the range of seemingly effortless motion and positioning, the better the mobility.

1.3.3.3 Speed

The faster the system can move a camera, the more options a film maker will have. Speed is the measure of how quickly the proposed design is able to move the camera from one point to another. The camera should be able to at least move as fast as the average cameraman could move a camera while maintaining a steady shot.

2 Design Overview

The design consists of four primary subsystems: the base, the tower, the camera head, and the control system. The base and tower are physically connected across a single interface, as are the tower and the camera head, and all three are utilized by a human operator through the control system.

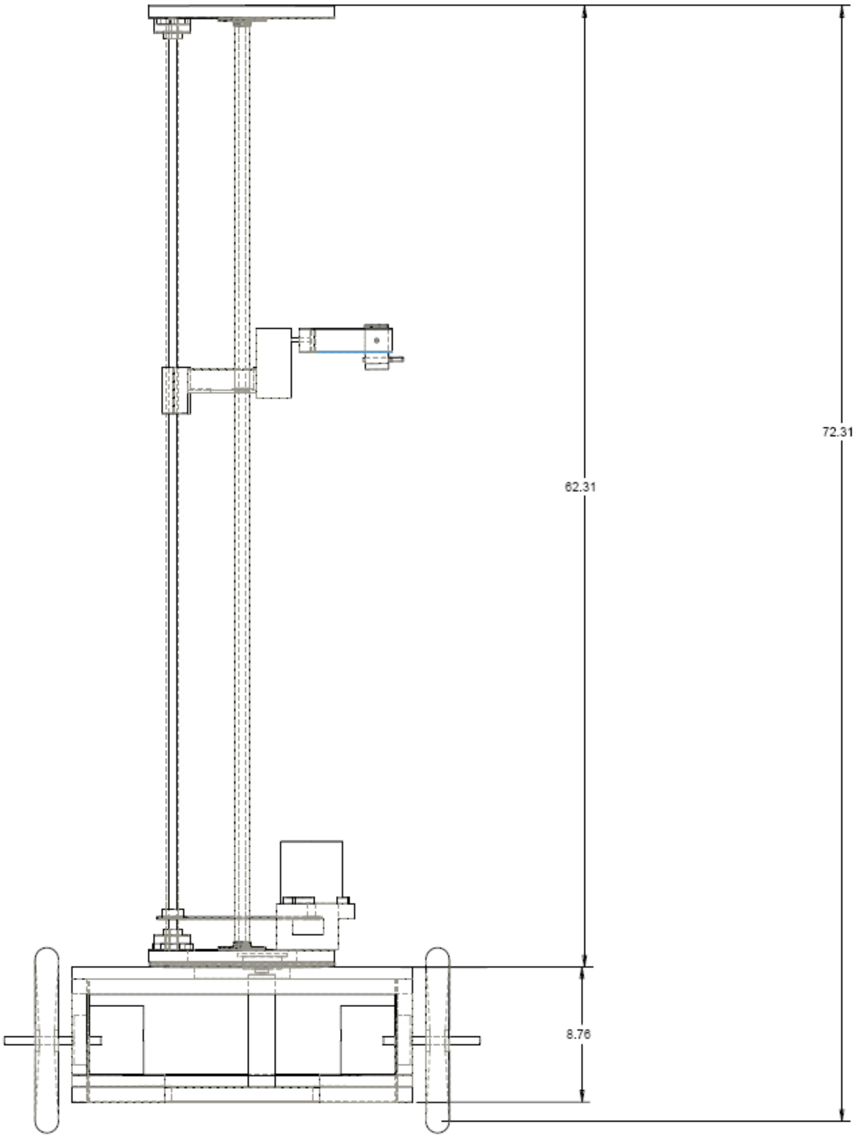


Figure 1: Basic Design

2.1 Base

The function of the base is to support the weight of the entire design, and to provide a physical foundation for all subsystems. Additionally, the base provides spatial motion of the camera in two dimensions (the dimensions constrained by the ground plane) by moving the entire system across the ground.

The base is a two-tiered design; the first tier supports the batteries, both drive motors, and the control electronics, and the second tier supports the tower subsystem.

The base is framed in 1”, 11-gauge square steel tubing welded together into one solid frame. Right-angles are utilized as much as possible to ease construction, except on the outside perimeter of the first tier; the perimeter of the first tier also uses 45° cuts, to implement an irregular octagonal shape. This shape is used to reduce the diagonal distance between opposing outside corners, which would otherwise require a larger surrounding free area for rotation of the base.

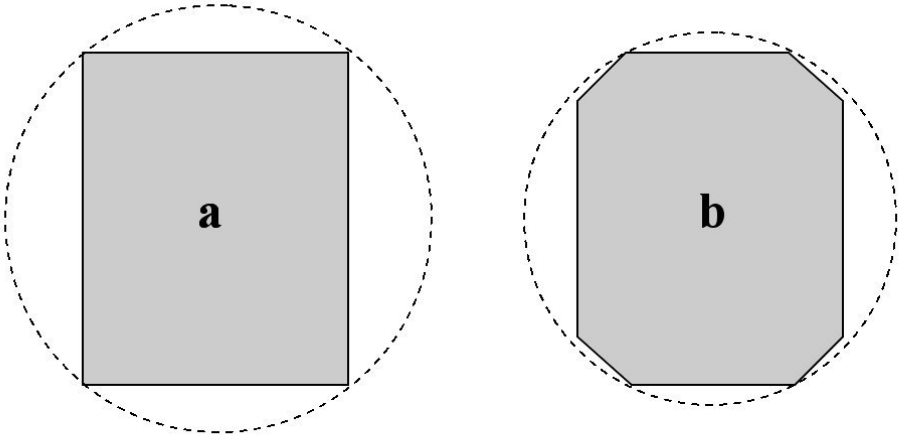


Figure 2: The rectangular base (a) requires a larger surrounding free area for rotation than an octagonal base (b) with equivalent length and width.

Each tier is also decked with a flat surface made of ¾” plywood. The plywood is bolted to the steel frame to ensure rigidity, and the addition of the plywood surfaces allows for easy installation of components with screws as well as providing lateral support to the steel frame. The deck of the first tier is made of two separate pieces that are bolted in separately; one solid piece would not be able to be removed because of the frame of the second tier, but because the division of the deck (lengthwise) is perpendicular to the supporting steel segments in the frame (which are mounted across the width), no structural strength is sacrificed in the two-piece implementation. The top deck is a single piece of plywood, with a central hole cut to mimic the hole of the turntable (elaborated further in the next section).

The drive motors are bolted to two steel plates, each of which is welded directly to the steel frame. The placement of the plates and of the bolt holes are such that when secured, the respective axes of the two motors are aligned with the center of the base. This allows the base to rotate in place by driving the motors in opposing directions at an equivalent speed, and because of symmetry, it results in no theoretical difference in system control between driving forwards or backwards. The motors themselves (as opposed to the location of each drive axle on the attached gearbox) are consolidated to one end of the base, so that the remaining space under the second tier can more effectively be used to house the control electronics.

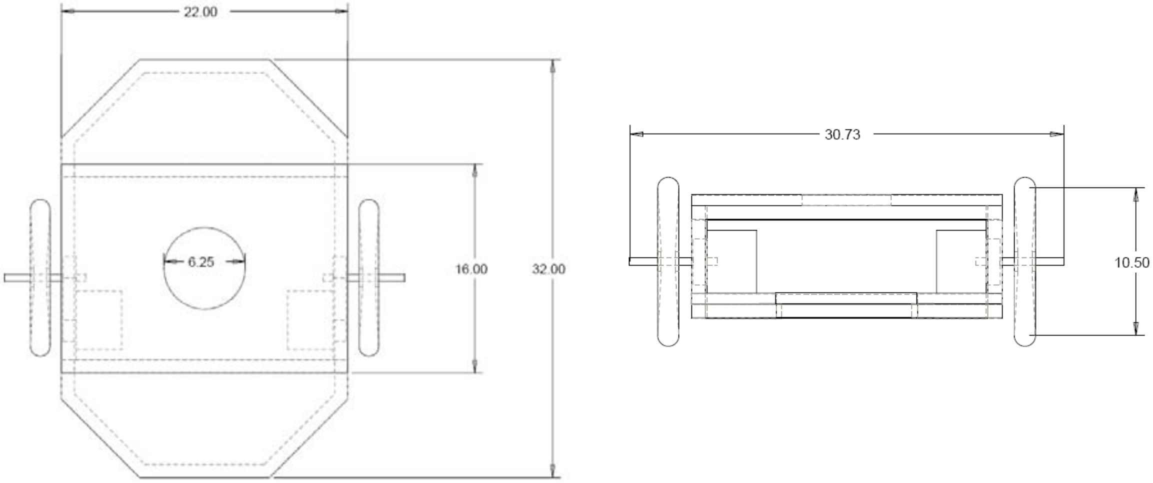


Figure 3: The basic construction of the base as well as location of wheel motors.

Finally, the base is mounted with a castor at either end, which is bolted to the underside of the first tier's plywood deck. The castors prevent the system from rotating about the axis created by the drive motors, and maintain both tiers parallel to the ground plane. By using a passive mechanical system of castors to accomplish this, there is no burden placed on the control system to balance the system, as there is with a Segway (which uses a similar drive system, but without the castor-balancing implementation).

2.2 Tower

The primary function of the tower is to facilitate camera movement perpendicular to the ground plane (changes in elevation), and to control camera orientation with three degrees of freedom.

The tower is bounded by two parallel discs, which are connected by an arrangement of three parallel rods (one threaded rod which turns, and two stationary support rods). The tower is connected to the second tier of the base through a turntable, which allows the entire tower to rotate independently from the base.

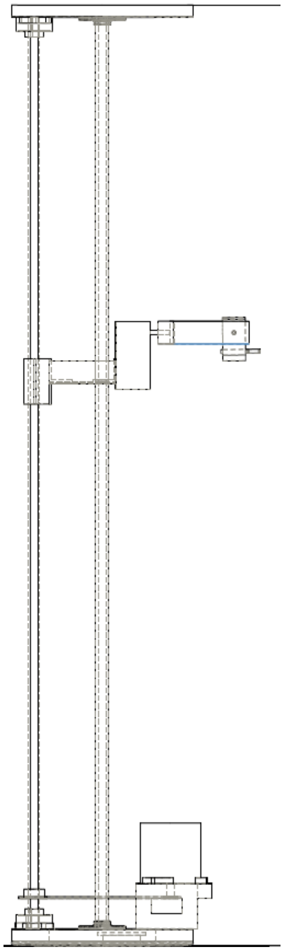


Figure 4: Tower Design

The turntable is the bottom of three layers which make up the lower tower disc. Above the turntable, a loop of chain is bounded by a perimeter of plywood to form the middle layer of the disc. The loop of chain is welded to a circular steel plate, which forms the top and final layer of the lower disc, and prevents the loop of chain from rotating within the plywood perimeter of the second layer. This construction is bounded together with bolts, and allows the turntable to be actuated by a motor-sprocket assembly; effectively, the bottom disc is a donut-shaped gear, with the gear teeth around the perimeter of the central hole. The chain is secured so that the inner diameter of the chain loop is approximately ¼” smaller than the inner diameter of the rest of the assembly, so that the connecting sprocket can more easily connect with the chain.

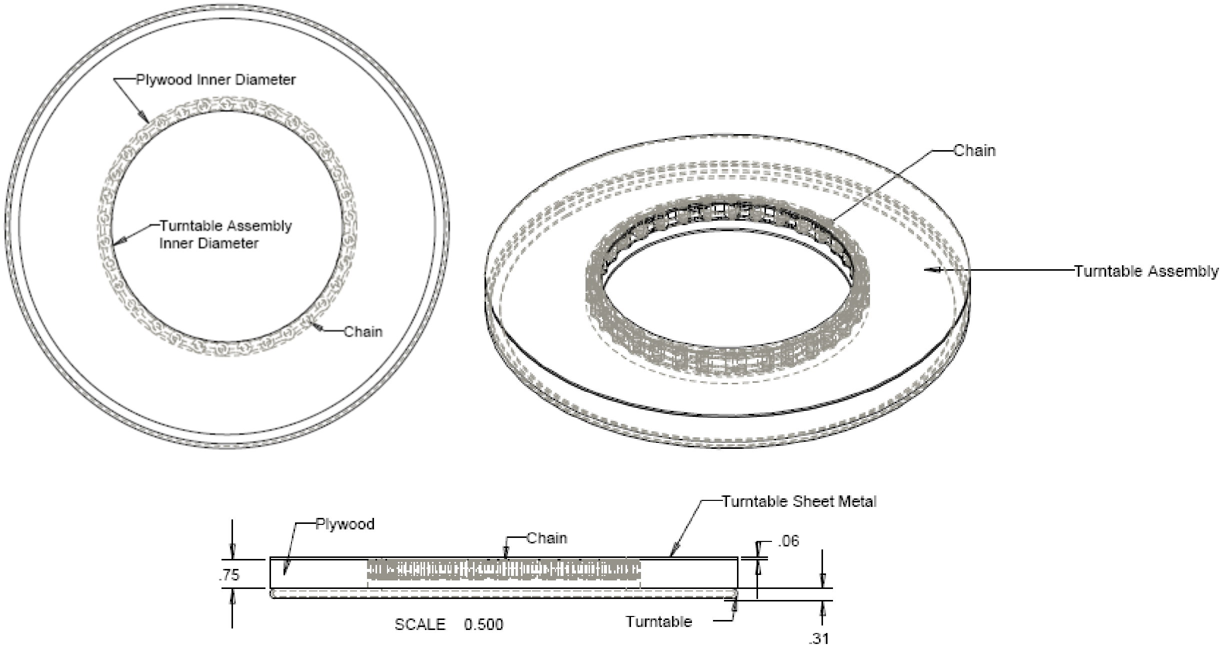


Figure 5: Turn Table Assembly

The actuating motor for the turntable assembly is a drill motor which is attached to the first tier deck of the base. The motor, with a corresponding sprocket on its 7/8 inch keyed shaft, is mounted vertically within the inner perimeter of the turntable assembly and is used to actuate rotation of the turntable. The rotation of the turntable effectively is the “pan” control of the camera.

The lower disc also is the mounting location for the 450-watt elevation-control motor. A motor mount is constructed out of welded 11-gauge steel tubing, which raises the motor off the turntable high enough to allow the motor sprocket to spin freely. This sprocket is chained to a corresponding sprocket on the threaded rod on the other side of the turntable, connecting the motor with the desired target of actuation.

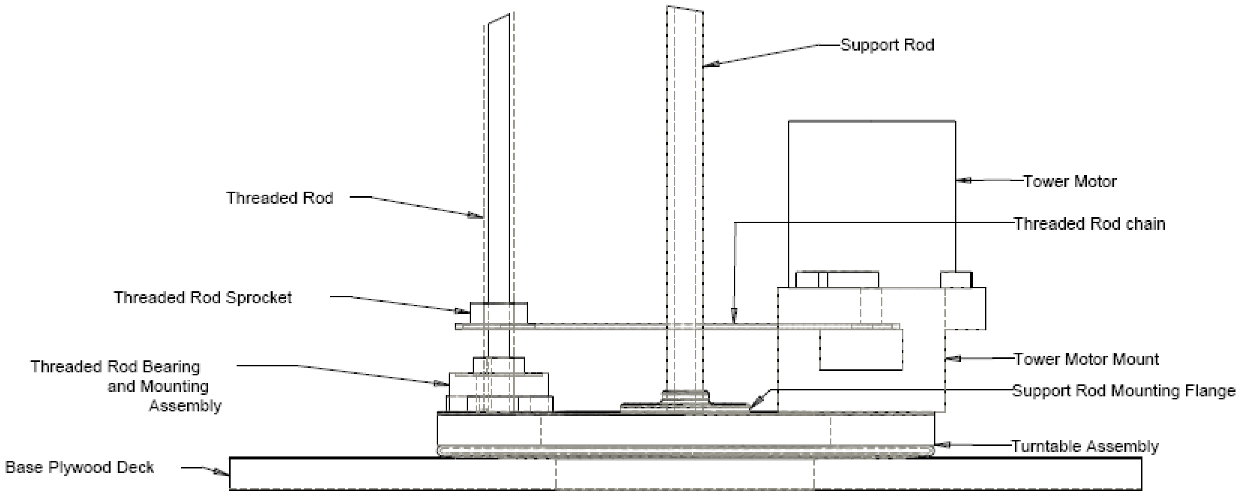


Figure 6: Motor Mount for Threaded Rod

The threaded rod is free to rotate along its axis, due to support from mounted bore bearings at either end of the rod, attached to the lower and upper disc assemblies. The threaded rod acts like a 5-foot-long worm gear, controlling the elevation of the attached platform nut (elaborated on further in the next section).

Additionally, two stationary support rods run in-between the lower and upper disc assemblies, and are connected at either end with threaded flanges. These support rods are intended to prevent torque on the top disc assembly, and to help distribute the weight supported by the top disc. The support rods also prevent the camera carriage from rotating about the threaded rod (which is explained in more depth in the next section).

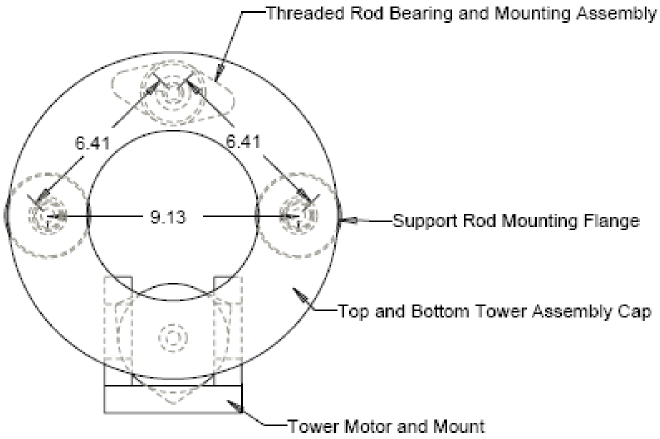

Threaded Rod Bearing and Mounting Assembly
6.41
6.41
9.13
Support Rod Mounting Flange
Top and Bottom Tower Assembly Cap
Tower Motor and Mount

Figure 7: Layout of Threaded Rod and Support Rods

The top disc assembly of the tower is simply a circular steel plate and a circular deck of plywood, to which is attached the bearing and both flanges corresponding to the threaded rod and both stationary rods, respectively.

2.3 Camera Head

The camera head assembly is attached to a carriage which rides a platform nut up and down the threaded rod as it turns. The carriage is a simple rectangular design made from welded segments of angle-iron; it is therefore very cheap, and very sturdy despite its low weight of only two pounds.

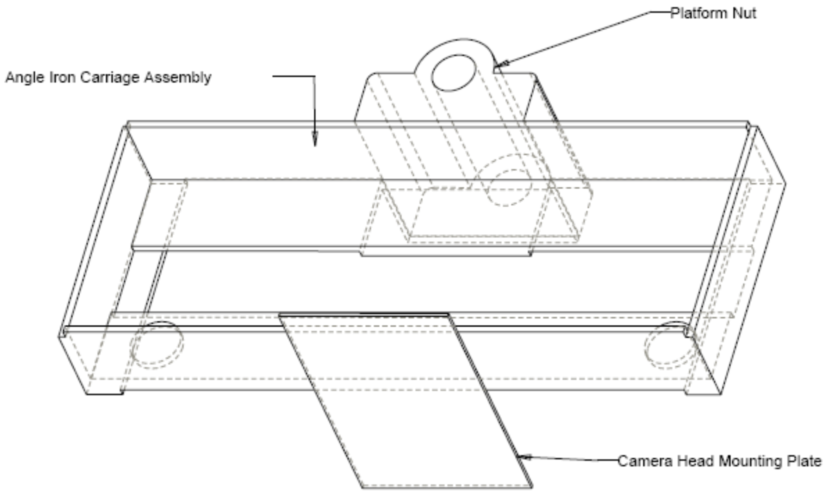


Figure 8: Carriage Construction

The carriage is bolted to the platform nut in four places, and has two 1” holes that allow the stationary rods to pass through the carriage; this prevents the carriage from rotating with the threaded rod (by the induced normal force against the carriage from both stationary rods), and insures that the rotation of the threaded rod results only in a change in camera elevation. The interior region of the camera carriage also serves as a supporting box in which to house the control electronics for the camera head.

The camera head is bolted in four places to the steel plate on the front of the camera carriage. The head has two servo actuators, which allow for control of the tilt and roll of the camera. The first servo (“roll”) rotates a yoke which holds the second servo, and that servo (“tilt”) rotates an embedded yoke that supports the camera. The entire camera head assembly, except for the housing of the roll servo, is made of pre-cut plastic and held together with screws. The housing of the roll servo is constructed of prefabricated aluminum.

2.4 Control System

The function of the control system on the robotic camera operator is to process commands given by the user and translate them into movements of the various actuators. Commands are given to the system via the use of a gamepad borrowed from a consumer gaming system. The commands from the gamepad are interpreted and commands are passed along to the appropriate microcontrollers. These microcontrollers process the commands into the various methods of controls used by the actuators. It is a simple, open loop control system designed to provide basic functionality for the robot.

2.4.1 Construction

During the development phase of the control system, it is constructed on a breadboard that allows easy relocation and changing of components. However, once the design is finalized, a breadboard is not robust enough to construct reliable circuitry. Circuits on breadboards have a tendency to be unreliable when subjected to the vibrations associated with being on a robot. Therefore, all components are soldered to a prototyping board. This allows flexibility as components changes can still be made, while also eliminating problems with loose components caused by vibration.

2.4.2 Architecture

The architecture of the control system of the robot is a system of microcontrollers arranged in a tree structure. At the top is the master microcontroller that communicates with a gamepad using a variation of SPI (Serial Peripheral Interface). The microcontroller sends a polling command to the gamepad and the gamepad responds with a digital signal that indicates the status of all inputs. The inputs are then interpreted by the master microcontroller, and passed on to the two lower level microcontrollers via the same SPI interface used for the game pad. Each of these microcontrollers processes the commands from the master microcontroller and generates commands for their respective actuators. A schematic for the overall architecture can be seen in Figure 2.

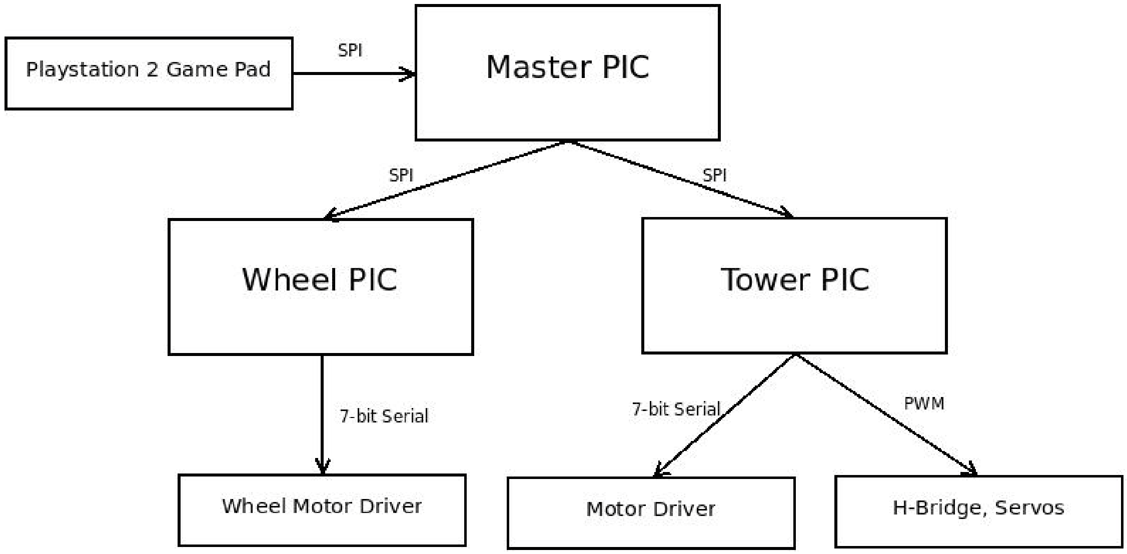


Figure 9: The overall architecture of the control system.

2.4.3 Human Interface

There are two major components to the human interface of the robot. The first and most important is the dead man's switch. This switch is a commercial switch produced for the same purpose in the marine industry. It has a tether that attaches to the operator at one end, and a switch at the other. If the tether becomes detached from the switch at any point in time, the switch changes state. It is connected in such a way as to cut off power to the moving parts of the robot when the switch changes state. The need for this switch was decided upon when the destructive potential of the robot due to its weight and power was realized.

The second human interface device is the gamepad. This device is used to control the motions of the robot. It was chosen for three reasons. First of all, it has lots of buttons and two joysticks, allowing for many options in control of the robot, and for future expansion. Additionally, the protocol for this device is well documented [2] so it is not difficult to connect to the selected microcontrollers. Finally, it is a very intuitive interface to many people, which saves a considerable amount of time and money in designing an interface specific to this project.

2.4.4 Actuator Control

There are three main ways that actuators are controlled on the robot. The wheel motors and the motor for the threaded rod of the tower are controlled using commercially available motor drivers. The actuators on the camera head are standard hobby servos, and are controlled using the variation of PWM (Pulse Width Modulation) characteristic of that type of actuator. Finally, the motor that actuates the rotation of the turntable is controlled using an H-bridge and the PWM function of the microcontroller.

2.4.4.1 Wheel and Threaded Rod Driver

These drivers are designed to provide the large amount of power required for these actuators. This driver accepts input via a 7-bit, even parity serial (RS232) signal. The difficulty in setting up communications between the microcontrollers and the motor drivers drivers is that the microcontroller serial port is setup for 8 or 9 bits without parity by default, and no option for 7-bit communication. In order to achieve 7-bit serial communication with a microcontroller that defaults to 8-bit communication, some knowledge of serial communications and binary numbers is required. The most significant bit of the 8-bit serial signal is always zero when sending data between 0 and 127, or 0x00 and 0x7F in hexadecimal. Therefore, 7-bit serial communication can be used for data in this range. The addition of the parity bit to the 7-bit signal makes the sum of the number of bits for each piece of data sent 8-bits. The parity bit takes the place of the most significant bit in what would ordinarily be the 8-bit signal. This is done by counting the number of 1's in the transmission, and setting the parity bit (formerly the most significant bit of the 8 bit signal) to high if they are odd, or low if they are even. This effectively makes every transmission have an even number of bits. This is the only real change needed to send a 7-bit signal.

The commands sent to the motor drivers are sent as follows. First, a '!' character is sent, alerting the controller that a new command is coming. The next character is either an 'A' or a 'B'. This informs the controller which channel the command is for. If the character is lowercase, that channel is to be reversed. Next, two characters that are the hexadecimal representation of an unsigned 8-bit number that indicate the value the channel should be run at are sent. Finally, an ASCII carriage return character is sent, signaling the end of the command.

The motor drivers have several different modes of operation. In the case of the driver for the threaded rod motor, it only has a single channel and has none of the more advanced modes. The driver for the wheels, on the other hand, has two modes of operation that are relevant to this system. The first mode is that each channel (and therefore each wheel) is controlled independently. This is useful on a system that does not steer using a difference of speed between the wheels. The second mode is mixed mode, where the controller controls the two channels together. In this mode, channel A specifies the speed of both wheels, and channel B specifies the difference in speed between them. This makes it easy to implement steering on a system such as this one, and is the mode used in this design.

2.4.4.2 Servo PWM

The servos that actuate the camera head are controlled using a standard RC servo signal. This signal is a form of PWM, although it operates at a much slower frequency than the PWM used for an ordinary motor. This PWM generates a signal with a 20 millisecond period, and varies the high time of the signal between about 1 millisecond and 2 milliseconds. The high time of the signal is what tells the servo what position it should go to.

In order to produce a PWM signal appropriate for a hobby servo, the period of the signal is divided up into ticks. The length of each tick is determined by the clock frequency of the microcontroller. Next, the necessary pulse width is determined by correlating the length of the pulse to a number between 0 and 255 that corresponds to the desired position of the servo. Upon completion of that calculation, the output to the servo is set high and a timer is set to count off the number of ticks for the high portion of the signal. Once this counter reaches its highest possible value, it sets an interrupt. The handler for this interrupt calculates the number of ticks for the low portion of the signal, and sets the output low and then sets the timer again.

2.4.4.3 Turntable PWM

The motor for the turntable is controlled using the PWM system built into the microcontroller. This system generates a PWM signal of desired frequency and duty cycle, which is then fed into an h-bridge, which provides the power necessary to drive the motor. The speed of the motor is controlled by varying the duty cycle of the PWM signal outputted the microcontroller. The larger the duty cycle, the more power the motor gets and the faster it spins.

2.4.5 Software

All software used on the microcontrollers is written in C. The compiler used for compilation of the C code is the Lite version of Hi Tech C.

3 Testing Methods

In order to determine how successful the design is, it needs to be subjected to a series of test that can determine its limits of operation, and prove that the system itself works.

3.1 Mechanical System

The testing of the mechanical system will ensure that everything moves as it should, when it should, and that the physical aspects of the system will not fail during operation.

3.1.1 Wheel and Threaded Rod Motors and Driver

The motor driver is supplied with power; the software package supplied with the driver is used to configure it. This software has ready-made testing features and even monitors and plots the magnitudes of the outputs to the two channels. To test the driver, it is connected to the serial port of the computer and observations are made to ensure that each channel achieves 90% of the maximum speed.

Before the motors themselves are mounted to the frame, it is important to check that they can be driven properly by voltages within their rated range, and more importantly, that the two motors produce the same results. A brief test is performed on each motor to ensure that between zero and the 90% of the maximum rated speed it does not exhibit unusual behavior such as overheating. This test is performed with a wheel attached to make the rotational behavior more visible. This test's success is dependent on the judgment of the tester and is essentially a check for defective behavior. The same tests are performed for the threaded-rod motor, as the driver is the same but one-channel.

3.1.2 Strength of the Base

When the motors and wheels are attached to the base, but before the tower is mounted, a test is performed to ensure that the base can support the weight of the tower. Weights in the amount of 200% of the tower weight are placed on the base and each motor is driven with a voltage that increases from zero until the entire system is moving at a swift walking pace, between 4 mph. This test is considered a success if no excessive stresses are apparent on the frame itself, or on the connections between the motor and frame, and if the motors can move the system at the desired speed.

3.1.3 Driver for Turntable motor

The driver is supplied with power and a test command signal is given to it. The pulse width modulated (PWM) output of the driver is viewed on an oscilloscope and compared with the expected values. The signals checked are full reverse, half reverse, stopped, half forward, and full forward, to verify that the PWM output operates linearly with duty cycles of within 5% of their expected values, and to verify that the polarity is reversed between forwards and backwards.

3.1.4 Turntable System

The motor and driver are connected to the turntable through a sprocket. The turntable has the tower attached to the top and the camera carriage is positioned at its maximum height. On the carriage, extra weight is placed to verify that the rotation does not cause the robot to tip. The motor's speed is slowly increased to verify that the turntable does indeed rotate, and to find the maximum operating speed to ensure a stable rotation without tipping. The tester, from a standstill, and using the maximum operating speed, moves the tower one full rotation. If this happens in less than 4 seconds, the system is a success.

3.2 Control System

The control system is tested from the bottom up, one subsystem at a time. As subsystems are verified to be working, they are combined and then tested as a complete unit. This method allows easier debugging by isolating problems to as few candidates as possible.

3.2.1 Servos

In order to test the function of the servos, the microcontroller is programmed to output signals that move the servos through a specific and easy to watch series of motions. The servos are then connected to the microcontroller and the result is observed. If the observed movements are the desired movements, then servo control is working correctly.

3.2.2 Wheels and Threaded Rod

The wheels and the threaded rod are using controllers from the same family by the same manufacturer; the controllers have the same operation and can be tested using the same methods. The testing of the wheel related software is primarily a test of the communications between the microcontroller and motor drivers. First, the microcontroller is programmed to output a valid set of commands to its serial port. The microcontroller is then connected to a computer running the RoboRun software, which is designed to communicate directly with the motor drivers. It uses the same protocol as the motor driver, so it is a good place to check whether or not the commands being sent by the microcontroller are valid or not. The RoboRun software includes a serial console that prints out all commands that are received. If the microcontroller is sending the correct commands using the correct communication method, they are printed in the console. If the wrong commands are being sent (i.e., 7-bit serial is not working properly), the commands show as garbage in the serial console. Once the commands have been verified as correct, the microcontroller can then be connected to the motor drivers. The microcontroller is programmed to output a series of commands that command the motors to move in a specific manner that is easy to watch. If the movements of the wheel motors match those being commanded, the test is successful.

3.2.3 Turntable

The function of the PWM software is tested by connecting the PWM output of the microcontroller to an oscilloscope and measuring the frequency and duty cycle of the outputted signal. If this signal is the one that is desired, and the duty cycle varies correctly when commands are sent to the microcontroller, the turntable subsystem is in working condition.

3.2.4 Gamepad

The testing of the Gamepad is accomplished using a microcontroller, a personal computer, and a program called PuTTY. For this system, the microcontroller is programmed to read data from the gamepad, and then output the state of each button over RS232 serial to the computer. This generates output in the PuTTY program that can be easily read by the person doing the test. When all buttons are reading correctly and the readings of the joysticks respond in a reasonable manner, the gamepad is working correctly.

3.2.5 Microcontroller Communication

The communication between the various microcontrollers is tested in a fairly simple way. First, the master microcontroller is programmed to take input from two switches and then send commands to the two slave microcontrollers. Each of these microcontrollers is setup with an LED light on an output and change its state depending on commands sent from the master controller. When communications are working correctly, operating the two switches connected to the master microcontroller will change the states of the LED's connected to the slaves. When this is behaving as specified, microcontroller communication is said to be working.

4 Testing Results

An analysis of the testing results will determine how successful the design is. The constraint concerning the schedule resulting in a lack of applicable testing results. The system was unable to be completed in accordance to the design specifications thus complete testing was unable to be performed.

4.1 Wheel Motor Testing

The wheel motor driver achieves the desired speeds for each channel, which is verified qualitatively by observation and also quantitatively using the driver software package. The wheel motors themselves are observed to function properly and are not defective. The servos on the camera-head function properly out-of-box, and are controllable as desired through their PIC.

4.2 Durability Testing

Being made from 11-gauge square steel tubing, the frame of the robot is much stronger than required and an excessive weight in the amount of 300 lbs is placed on the frame, causing no visible deformities, and without affecting movement.

4.3 Other Motor Testing

Due to the turntable motor assembly being incomplete, neither the motor nor the driver are tested. This is also the case for the threaded-rod motor.

4.4 Control System Testing

The control system is not completely constructed due to time constraints, and cannot be completely tested. Of the components that are complete, the servos move exactly as they should when commanded to, and they do not move when they are not commanded to. The operation of the servo control is successful. The wheel system works correctly. It operates in mixed mode, and the speeds of the motors vary correctly with the commands sent to them. The motor and driver for the threaded rod are not tested, and the turntable system is not tested.

4.5 Game Pad Testing

The outputs of the game pad are correct, when a button is pressed on the pad, the correct button is shown as being pressed on the output to the computer. The joysticks also report values, but they do not work as expected. First, when the user releases the joysticks and they physically return to their center positions, it is not reflected in the output. Instead, the values of their last position are reported. The second issue is that the position of each joystick reported by the gamepad is not linear, which makes them undesirable for the intended use. The SPI communication of the microcontrollers is not complete, and so it is not tested.

5 Conclusions

Though the prototype was never fully completed, it was possible to arrive at some conclusions about the design and how it stacked up against what we planned to create at the beginning of the year, as well as how the project itself was managed.

5.1 Performance against constraints and criteria

Without the completion of the control system, it is difficult to acuratly gauge how well the design stands up to our criteria, however some aspects of the design can still be at least partially evaluated

5.1.1 Performance

Mechanically the system should meet the constraint of altitude range; the lower height of the tower is 2 feet above the ground and the highest point is at approximately 6 feet above the ground. The pan, tilt, roll, and rotation mechanisms of the system serve to simulate the movement a human operating a hand-held camera.

5.1.2 Sound

Though the whole system was never operated together, the individual motors had all been run independent of the system. Each of these motors made a significant amount of noise when they were being actuated. As there was no plan to put any form of sound dampening in place, it can be assumed that it would be equally as loud in the finished product. However, given the correct sound recording set up, it still should be quiet enough to film with under most conditions.

5.1.3 Set up

Assuming the battery is charged, the only setup required for a shot is to mount the camera, turn the system on and wait for the software to load. This is significantly shorter than most other systems and defiantly would be expected to be shorter than the one hour stated in the design criteria.

5.1.4 Mobility

As far as the mechanical design goes, there are no limiting factors that could hamper the system's ability to successfully and easily navigate a 25 by 25 foot sound stage, with the camera between 2 and 6 feet. However as this was never actually attempted due to an incomplete control system, this cannot be verified.

5.1.5 Durability

The entire system had been disassembled and reassembled completely on numerous occasions with no adverse effects. The only component to fail was the turntable motor, however this was due to a deviation from the original design that was forced by scheduling issues, and would not be expected to have occurred if it was assembled properly.

5.1.6 Usage, Precision, and Speed

These remaining criteria cannot be adequately compared to the system without completion of the control system. Upon completion of the control system the testing of these remaining criteria can be performed and the final system performance assessed more accurately.

5.2 Budget and Schedule

The total purchases fell within the allotted budget. Out of the $2700.00 allotted to the project, purchases for the Robotic Camera Operator totaled $2660.50. The design's most expensive purchases were the motors, the threaded rod and platform nut, and the motor controllers.

Time was perhaps the greatest setback for the group. Each task required a lot more time than was allotted in the original schedule. The time necessary for programming of the PICs was greatly underestimated; becoming familiar with the PICs and the PIC programming language proved more difficult than expected for the simple reason that no one had previous experience with them. Mis-orders and lost packages added further to the difficulties with scheduling.

5.3 Recommendations

If this project was to be continued, it would be desirable to complete the control system so that an operator could, using open loop control, easily navigate the robot around. Once this is completed, the encoders could be added to the system as well as some memory so that a teach and repeat system could be easily implemented.

6 Bibliography

[1] Motion Control Rigs, Cameras, Heads, Platforms, Pan Tilt Heads, Cranes, and the Home of Milo. 26 Apr. 2009 <http://www.mrmoco.com/>.

[2] "Playstation controller interfacing -." Hack a Day. 26 Apr. 2009 <http://hackaday.com/2008/03/07/playstation-controller-interfacing/>.

A Control System Schematics

This section contains schematics for the control system. It should be noted that the items depicted in Figure C-1 are also included with all of the other microcontrollers, but are not shown on those schematics.

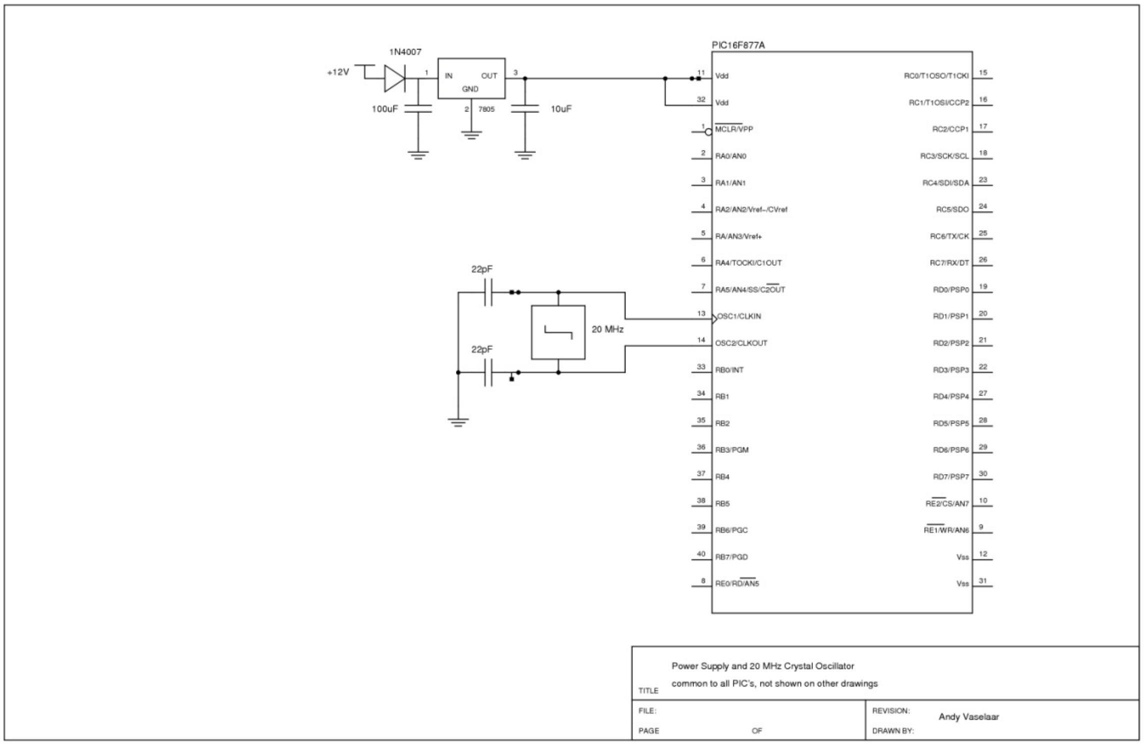


Figure A-1: Schematic for items attached to all microcontrollers.

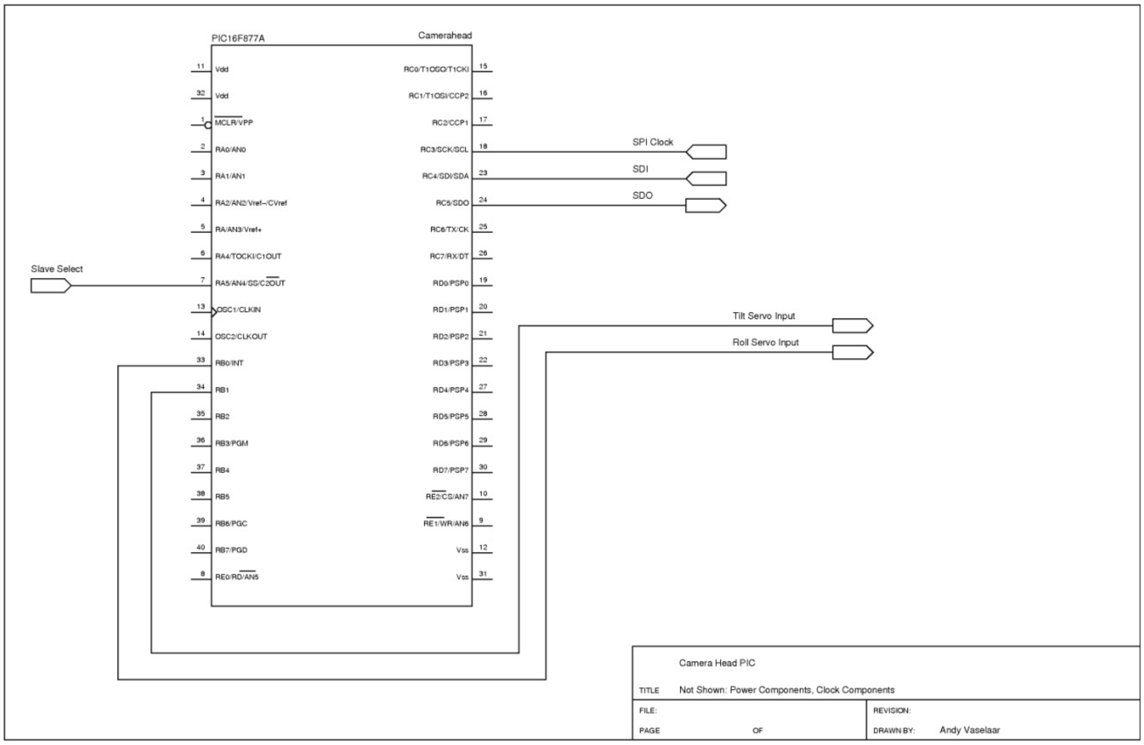


Figure A-2: This schematic depicts the electrical setup of the Camera Head microcontroller.

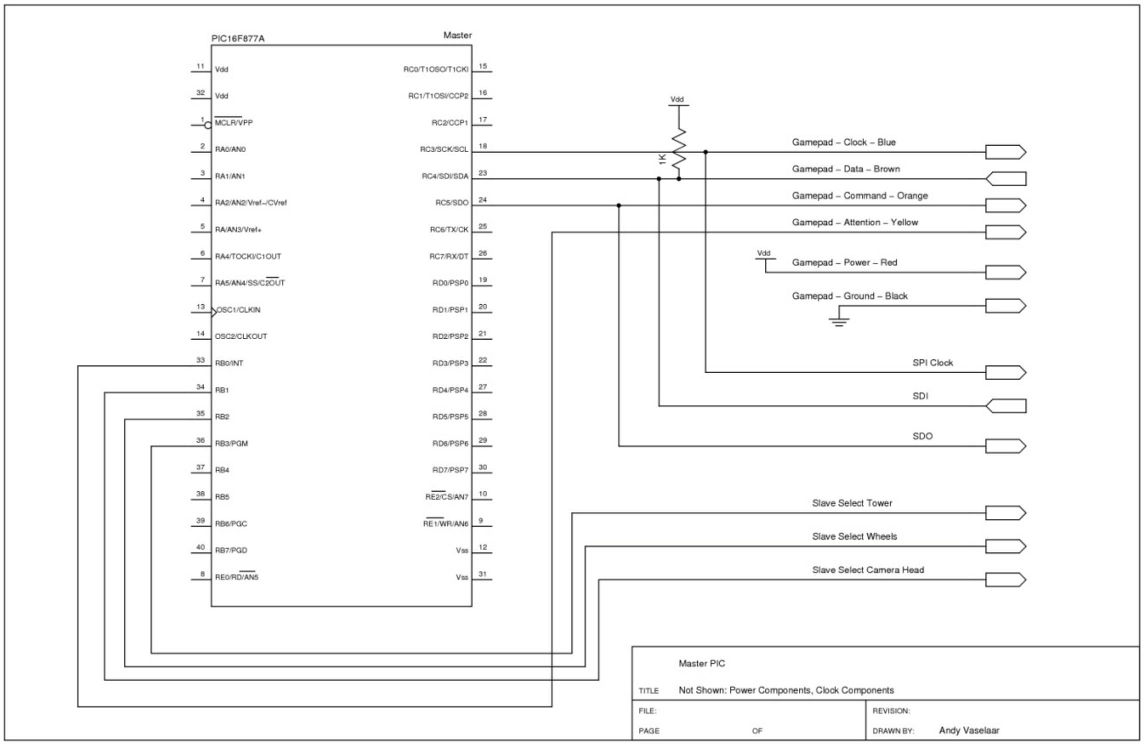


Figure A-3: This schematic depicts the electrical setup of the Master microcontroller.

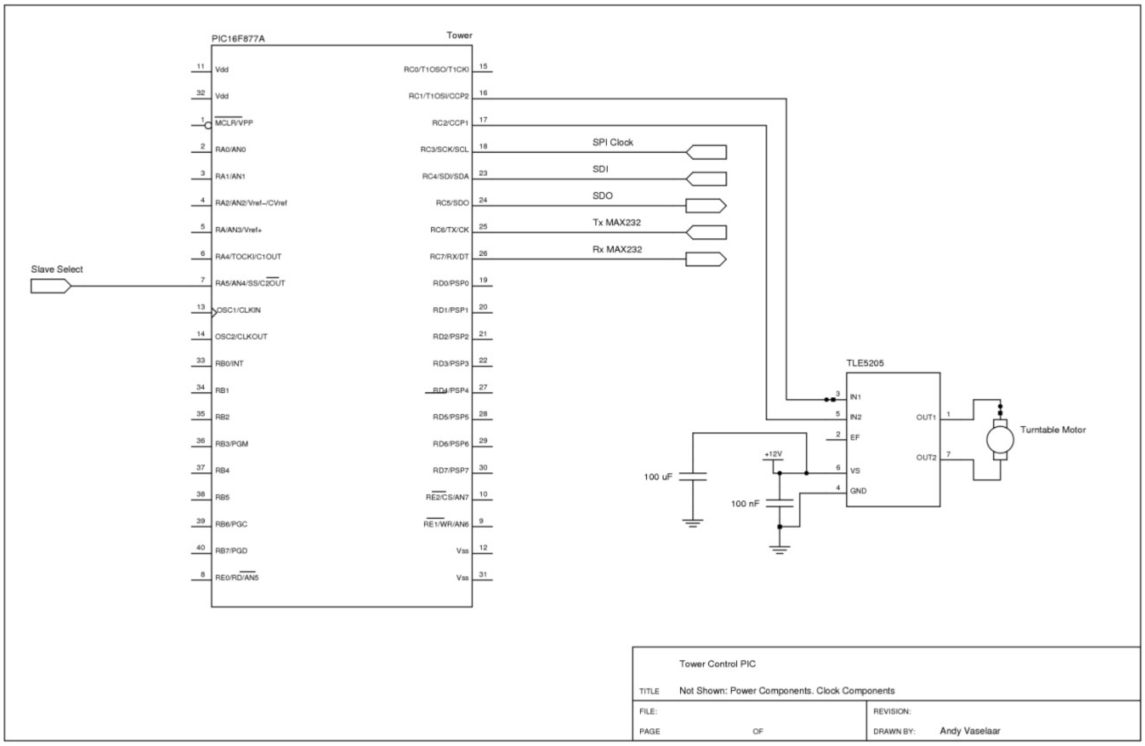


Figure A-4: This schematic depicts the electrical setup of the Tower Control microcontroller.

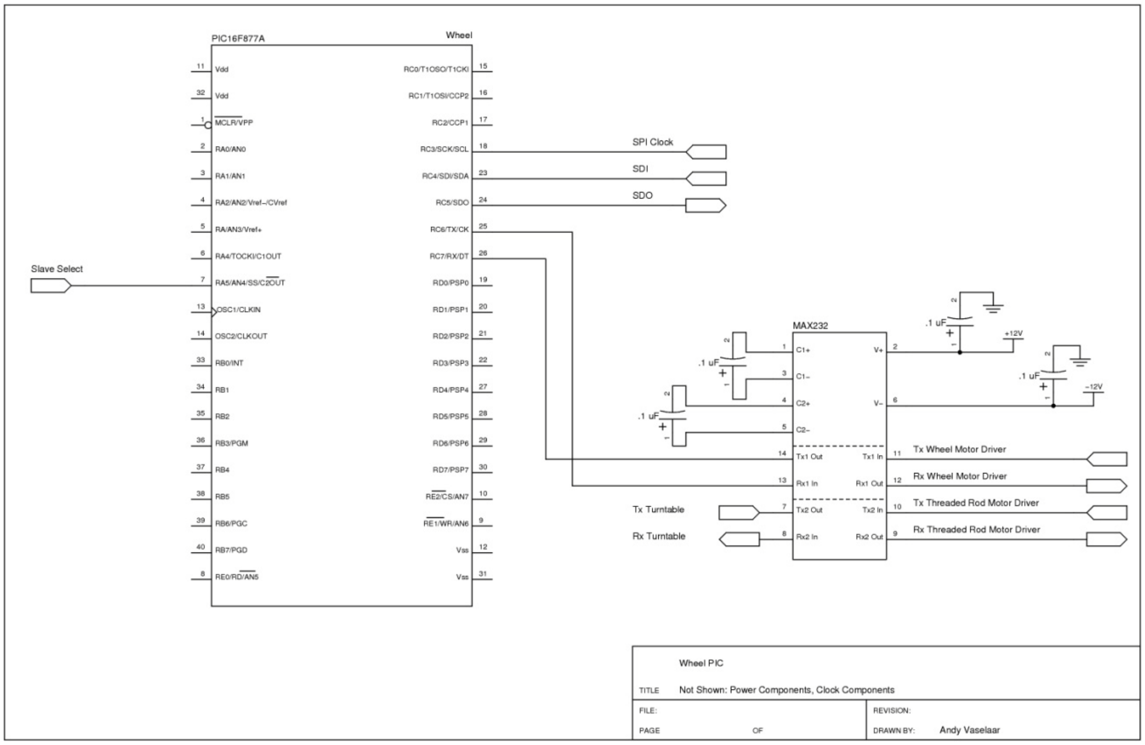


Figure A-5: This schematic depicts the electrical setup of the Wheel microcontroller.

B Code

This Appendix contains all of the C code for the system. It is to be compiled with HiTech C. Additional include files have been excluded.

Code For the Master microcontroller

(also includes wheel control algorithms)

```
/*Robot Camera Operator
 * Master PIC

 PIC Pin Connections:

 SPI:
 Pin 23 - SDI Gamepad Brown, SDO pin 24 on Slave PICS
 Pin 24 - SDO - Gamepad Orange, SDI Pin 23 on Slave PICS
 Pin 18 - SPI Clock - Blue Gamepad, Pin 18 on slave PICS

 Serial:
 Pin 25 - USART TX
 Pin 26 - USART RX

 Slave Select Lines
 Pin 33/RB0 - Gamepad - Yellow - Attention
 Pin 34/RB1 - TowerPIC Pin 7
 Pin 35/RB2 - WheelPIC Pin 7
 Pin 36/RB3 - HeadPIC Pin 7

 Other Connections:
 Gamepad - Grey - vibration motor power (7.2 - 9V?)
 Gamepad - black - ground
 Gamepad - red - power - 3.3 V, should work at 5V as well.
 Gamepad - Green - acknowledge -  unused (tells the playstation that the
controller is there)

 Serial Settings:
 9600 Baud, N,8,1, no flow control
 */

//------------ INCLUDES ------------------------------- //
#include <htc.h>
#include "main.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "usart.h"
//#include <delay.c>

//------------ DEFINES ------------------------------- //

// this is a macro shortcut (this code will replace bitrev(c) at compile
time)
// for reversing the bit order
#define bitrev(c) c = (c & 0x0F) << 4 | (c & 0xF0) >> 4; \
                              c = (c & 0x33) << 2 | (c & 0xCC) >> 2; \
                              c = (c & 0x55) << 1 | (c & 0xAA) >> 1;

//------------ CONFIG ------------------------------- //
//Disable Watchdog Timer
//Setup High Speed Crystal Oscillator
//Disable Brownout REset
//Unprotect memory?
__CONFIG(WDTDIS & HS & BORDIS & UNPROTECT);

//Define Crystal Frequency

#ifndef _XTAL_FREQ
 // Unless already defined assume 4MHz system frequency
 // This definition is required to calibrate __delay_us() and __delay_ms()
      #define _XTAL_FREQ 20000000
#endif



/***** Ports *****/
#define pin_GP                RB0 // slave select for the gamepad
#define pin_T                 RB1 // slave select for Tower PIC
#define pin_W                 RB2 // slave select for the Wheel PIC
#define pin_H                 RB3 // slave select for Head PIC



//------------ Prototypes ------------------------------- //
void Init(void);
void poll_controller(void);
// send and receive a byte from the controller via SPI
void ps2_txrx(unsigned char data_out,unsigned char *reg_in);
void putsUART( char *data);
void output_buttons();
void process_commands();
void transmit_instructions(unsigned char data_out, unsigned char *reg_in,
unsigned char uc);

//Move to Wheel PIC
void sevenbitsend(unsigned char byte);
void send_command(char dir, char val1, char val2);
void wheel_control();
int robo_abs(int number);
unsigned int bitCount(unsigned char n);
void wheel_control2();

//------------ Global Variables ------------------------------- //

int read_buf_index =0;
unsigned char print_buf[80];
unsigned char print_buf_index = 0;
```

```
uint8 PS2_ID, MSG;
uint8 PS2_byte1, PS2_byte2, PS2_byte3, PS2_byte4, PS2_byte5, PS2_byte6;
uint8 PS2_prev_byte1, PS2_prev_byte2, PS2_prev_byte3, PS2_prev_byte4,
PS2_prev_byte5, PS2_prev_byte6;
unsigned char PS2_pressure[12];

int RJX,RJY,LJX,LJY;
int count=0;
int LJYloc = 0 , RJXloc = 0, LJXloc = 0 , RJYloc = 0;

// ---------------  utility functions ---------------- //

void ps2_txrx(unsigned char data_out,unsigned char *reg_in){
      bitrev(data_out);  //bit reverse the data
      SSPBUF = (data_out); //write the data to the outgoing buffer
      while(!BF); //wait for incoming data to fill the incoming buffer
      *reg_in = SSPBUF; //write incoming data to *reg_in
      //bitrev(*reg_in); //reverse the bits of reg_in
}


void print_ascii_r(unsigned char t, unsigned char t2)
{
      // there's not enough time to send out serial RS-232 back to the
computer
      // in between every byte, so output is stored in a buffer until the end
      // of a packet.
      print_buf[2*print_buf_index] = t;
      print_buf[2*print_buf_index+1] = t2;
      if (print_buf_index < 38)
            print_buf_index++;
}

void putsUART( char *data)
{
  do
  {    // Transmit a byte
    putch(*data);
  } while( *data++ );
}


void
main(void)
{
      Init();  //initialize communications, gamepad, etc.

      while(1)
      {
            poll_controller();      //Get states of controls from controller
            //process_commands(); //decide where to send which, and do it
            wheel_control(); //test wheel control function, will be moved to
wheel pic
      }
}
```

```
void
Init(void)
{
      unsigned char response;

      //Setup Slave Select Line TRIS
      TRISB0=0; //Gamepad
      TRISB1=0; //Tower
      TRISB2=0; //Wheel
      TRISB3=0; //Head

      //Setup USART
      INTCON=1;   //disable all interrupts
      init_comms(); //Initialize the USART

      //Setup SPI
      TRISC5 = 0; //Clear TRISC5 bit to enable SDO for SPI (p75)
      TRISC3 = 0; //SCK Master Mode
      TRISC4 = 1; //Slave Select enabled
      SSPSTAT = 0b00000000;
      SSPCON = 0b00110010;
      // Fosc = 20 MHz, so the SPI data rate is 312.5kHz.
      // CKP =1 (idle clock state is high, active low)
      // CKE=0    data is read when the clock transitions
      //from active to idle.  This causes the clock
      //to start out going low, and then high mid-cycle.

      sevenbitsend('!');  //send out in seven bit serial glory.
      sevenbitsend('A');
      sevenbitsend('0');
      sevenbitsend('0');
      sevenbitsend(0x0D);
      //sevenbitsend(' ');
      /*sevenbitsend('!');
      sevenbitsend('B');
      sevenbitsend('0');
      sevenbitsend('0');
      sevenbitsend(0x0D);*/
      char hexnum[2];
      itoa(hexnum, 3, 16);
      send_command('B', hexnum[0], hexnum[1]);
}
void poll_controller()
{
      unsigned char response;
      char buf[20];
      char t;

      RJX = 0;
      RJY = 0;
      LJY = 0;
      LJX = 0;
      for(int i=0; i <=100; i++)
      {
      pin_GP = 0;                         // pull ATT low to signal gamepad
      ps2_txrx(0x01, &response);                           // send:    Start
```

```
(0x01 LSBF (order flipped in ps2_tx)
      ps2_txrx(0x42, &PS2_ID.byte); // send:    Request Data (0x42 LSBF)

      ps2_txrx(0x00, &MSG.byte);// receive gamepad ID:      should be '0x5A'
for 'ready to send'

      ps2_txrx(0x00, &PS2_byte1.byte);                // receive: 1st byte of
button states and set big (left) motor to Left arrow pressure
      ps2_txrx(0x00, &PS2_byte2.byte);                // receive: 2nd byte of
button states and turn on small (right) motor if Right arrow pressure = 0xFF
      ps2_txrx(0x00,&PS2_byte3.byte);                 // receive: 3rd byte of
button states
      ps2_txrx(0x00,&PS2_byte4.byte);                 // receive: 4th byte of
button states
      ps2_txrx(0x00,&PS2_byte5.byte);                 // receive: 5th byte of
button states
      ps2_txrx(0x00,&PS2_byte6.byte);                 // receive: 6th byte of
button states

      ps2_txrx(0x00, &PS2_pressure[0]);
      ps2_txrx(0x00, &PS2_pressure[1]);
      ps2_txrx(0x00, &PS2_pressure[2]);
      ps2_txrx(0x00, &PS2_pressure[3]);
      ps2_txrx(0x00, &PS2_pressure[4]);
      ps2_txrx(0x00, &PS2_pressure[5]);
      ps2_txrx(0x00, &PS2_pressure[6]);
      ps2_txrx(0x00, &PS2_pressure[7]);
      ps2_txrx(0x00, &PS2_pressure[8]);
      ps2_txrx(0x00, &PS2_pressure[9]);
      ps2_txrx(0x00, &PS2_pressure[10]);
      ps2_txrx(0x00, &PS2_pressure[11]);
      pin_GP = 1;

      RJX = PS2_byte3.byte-127 + RJX;
      RJY = PS2_byte4.byte-127 + RJY;
      LJX = PS2_byte5.byte-127 + LJX;
      LJY = PS2_byte6.byte-127 + LJY;

      }

      RJX = RJX/100;
      RJY = RJY/100;
      LJX = LJX/100;
      LJY = LJY/100;
}

void
output_buttons()  {
      //Outputs the states of the gamepad buttons to the USART, meant to be
used
      //with PuTTY
      putch('\f');
      putch(MSG.byte);
      printf("\r RJX [%d]\n",RJX);
      printf("\r RJY [%d] \n",RJY);
      printf("\r LJX [%d]\n",LJX);

      printf("\r LJY [%d]\n",LJY);

      printf("\r Select: [%d] \n",PS2_Select);
      printf("\r L3: [%d] \n",PS2_L3);
      printf("\r R3: [%d] \n",PS2_R3);
      printf("\r Start: [%d] \n",PS2_Start);
      printf("\r Up: [%d] \n",PS2_Up);

      printf("\r Right: [%d] \n",PS2_Right);

      printf("\r Down: [%d] \n",PS2_Down);

      printf("\r Left: [%d] \n",PS2_Left);

      printf("\r L2: [%d] \n",PS2_L2);
      printf("\r R2: [%d] \n",PS2_R2);

      printf("\r L1: [%d] \n",PS2_L1);

      printf("\r R1: [%d] \n",PS2_R1);

      printf("\r Triangle: [%d] \n",PS2_Triangle);

      printf("\r Circle: [%d] \n",PS2_Circle);

      printf("\r X: [%d] \n",PS2_X);

      printf("\r Square: [%d] \n",PS2_Square);

      }

void
process_commands()
{
/* The function of this function is to take the commands from the gamepad,
 * and turn them into a standardized number that can be sent to each of the
individual
 * slaves
 */
unsigned char reg_in;  //will contain whatever the PICS send back, which is
probably nothing

//send data to the Wheel PIC
      transmit_instructions(LJX, &reg_in, 'W');
      transmit_instructions(LJY, &reg_in, 'W');
}

void wheel_control()
{
      /*Uses Mixed Mode on the driver*/
      unsigned char chan1dir, chan2dir;
      char numberValA[2];
      char numberValB[2];
      int turn_mag = 0;
      int drive_mag = 0;
      int turn = LJX;
      int drive = LJY;

      if(drive > 0)     //Determine direction
            chan1dir = 'a'; //Backward
      else
            chan1dir = 'A'; //Forward

      if(turn > 0)            //Changes the direction
            chan2dir = 'B';   //Right
      else
            chan2dir = 'b'; //Left

      drive_mag = robo_abs(drive);
      turn_mag = robo_abs(turn);

      turn_mag = turn_mag / 10;
      //drive_mag = drive_mag / 20;

      if(drive_mag > 127)
            drive_mag = 127;
      if(turn_mag > 127)
            turn_mag = 127;

      if(PS2_Circle == 0)
            turn_mag = 0;
      if(PS2_X == 0)
            drive_mag = 0;

      itoa(numberValA, drive_mag, 16); //convert to ASCII characters
      if(drive_mag < 16){
            numberValA[1] = numberValA[0];
            numberValA[0] = '0';
      }

      itoa(numberValB, turn_mag, 16);
      if(turn_mag < 16){
            numberValB[1] = numberValB[0];
            numberValB[0] = '0';
      }

      for(int x=0; x<=1; x++) //can't have a null character, need it to be a
0
      {
            if(numberValA[x]==0)
                  numberValA[x]='0';
            if(numberValB[x]==0)
                  numberValB[x]='0';
      }

      send_command(chan1dir, numberValA[0], numberValA[1]);
      sevenbitsend('\t');
      send_command(chan2dir, numberValB[0], numberValB[1]);
      sevenbitsend('\n');
}
```

```
int robo_abs(int number) {
      if(number>=0)
            return number;
      else return -1*number;

}

void wheel_control2()
{
      /*Uses Mixed Mode on the driver*/
      unsigned char chan1dir, chan2dir;
      char numberValA[2];
      char numberValB[2];
      unsigned int RJXloc_mag = 0;
      unsigned int LJYloc_mag = 0;

      //if(count%1000 == 0)
      //{
            if(PS2_L1 == 0 && LJYloc < 127) //Decrease Speed
                  LJYloc = LJYloc + 1;          //Do what we would do with a
joystick pointed down
            if(PS2_R1 == 0 && LJYloc > -127)//Increase Speed
                  LJYloc = LJYloc - 1;          //Do what we would do with a
joystick pointed up

            if(PS2_L2 == 0 && RJXloc > -127)//Increase Turning
                  RJXloc = RJXloc - 1;       //Do what we would do with a
joystick pointed left
            if(PS2_R2 == 0 && RJXloc < 127)//Decrease Turning
                  RJXloc = RJXloc + 1;          //Do what we would do with a
joystick pointed right
      //    count = 0;
      //}
      //count = count + 1;

      if( LJYloc > 0  ) //Determine
            chan1dir = 'a'; //Backward
      else
            chan1dir = 'A'; //Forward

      if(RJXloc > 0 )         //Changes the direction
            chan2dir = 'B';   //Right
      else
            chan2dir = 'b'; //Left

      LJYloc_mag = abs(LJYloc);
      RJXloc_mag = abs(RJXloc);

      //RJXloc = RJXloc / 20;
      //LJYloc = LJYloc / 20;

      /*if(LJYloc_mag > 127)
            LJYloc_mag = 127;
      if(RJXloc_mag > 127)
            RJXloc_mag = 127;*/
```

```
	if(PS2_Circle == 0)
		RJXloc = 0;

	itoa(numberValA, LJYloc_mag, 16); //convert to ASCII characters
	if(LJYloc_mag < 16){
		numberValA[1] = numberValA[0];
		numberValA[0] = '0';
	}

	itoa(numberValB, RJXloc_mag, 16);
	if(RJXloc_mag < 16){
		numberValB[1] = numberValB[0];
		numberValB[0] = '0';
	}

	for(unsigned int x=0; x<=sizeof(numberValA); x++) //can't have a null 
character, need it to be a 0
	{
		if(numberValA[x]==0)
			numberValA[x]='0';
		if(numberValB[x]==0)
			numberValB[x]='0';
	}

	send_command(chan1dir, numberValA[0], numberValA[1]);
	sevenbitsend('\t');
	send_command(chan2dir, numberValB[0], numberValB[1]);
	sevenbitsend('\n');
}

void send_command(char dir, char val1, char val2) {
	sevenbitsend('!');
	sevenbitsend(dir);
	sevenbitsend(val1);
	sevenbitsend(val2);
	//Return and commit command
	sevenbitsend(0x0D);
}

void sevenbitsend(unsigned char byte)
{
	//Input: unsigned character
	//Output: Converts to 7 bit serial, even parity, sends out USART
	//Depends: bitCount();
	unsigned int check;

	check = bitCount(byte);  //count the number of 1's
	//check = 2;
	if(check % 2 == 0)
	{
	// It's even
		putch(byte & 0x7F);//set first bit to 0
	}
	else
	{
      // It's odd
            putch(byte | 0x80); //set last bit to 1
      }
}

void
transmit_instructions(unsigned char data_out, unsigned char *reg_in, unsigned
char uc)
{
/*This function takes the standardized numbers from process_commands and
transmits them
 * via SPI to each of the slave PICs.
 */
      switch(uc){ //Set the slave select line for the PIC we're talking to
      case 'T':   //tower
            pin_T=0;
            break;
      case 'W':   //wheel
            pin_W=0;
            break;
      case 'H':   //camera head
            pin_H=0;
            break;
      default:
            break;
      }

      SSPBUF = (data_out); //write the data to the outgoing buffer
      while(!BF);             //wait for incoming data to fill the incoming
buffer
      *reg_in = SSPBUF;       //write incoming data to *reg_in

      switch(uc){ //set the slave select line back high again
      case 'T':
            pin_T=1;
            break;
      case 'W':
            pin_W=1;
            break;
      case 'H':
            pin_H=1;
            break;
      default:
            break;
      }
}

unsigned int
bitCount(unsigned char n)
{
  // This is for 32 bit numbers.  Need to adjust for 64 bits
  register unsigned char tmp;

  tmp = n - ((n >> 1) & 033333333333) - ((n >> 2) & 011111111111);

  return ((tmp + (tmp >> 3)) & 030707070707) % 63;
```

Additional Code for the Master Microcontroller

```
#ifndef _MAIN_H_
#define _MAIN_H_

#define Delay(x) DELAY = x; while(--DELAY){ Nop(); Nop(); }

/***** Buttons *****/
// these macros can be used as meaningful shortcuts into
// the response bytes in a PS2's return packet.
#define PS2_Select            PS2_byte1.bit0
#define PS2_L3                PS2_byte1.bit1
#define PS2_R3                PS2_byte1.bit2
#define PS2_Start       PS2_byte1.bit3
#define PS2_Up                PS2_byte1.bit4
#define PS2_Right       PS2_byte1.bit5
#define PS2_Down        PS2_byte1.bit6
#define PS2_Left        PS2_byte1.bit7

#define PS2_L2                PS2_byte2.bit0
#define PS2_R2                PS2_byte2.bit1
#define PS2_L1                PS2_byte2.bit2
#define PS2_R1                PS2_byte2.bit3
#define PS2_Triangle    PS2_byte2.bit4
#define PS2_Circle            PS2_byte2.bit5
#define PS2_X                 PS2_byte2.bit6
#define PS2_Square            PS2_byte2.bit7

#define PS2_RJX               PS2_byte3.byte
#define PS2_RJY               PS2_byte4.byte
#define PS2_LJX               PS2_byte5.byte
#define PS2_LJY               PS2_byte6.byte

/***** Types *****/
// the union lets us access the bytes coming back from a PS2
// either bit by bit or by a whole byte.  same memory location,
// just two different names.
typedef union {
  struct {
    unsigned bit7:1;
    unsigned bit6:1;
    unsigned bit5:1;
    unsigned bit4:1;
    unsigned bit3:1;
    unsigned bit2:1;
    unsigned bit1:1;
    unsigned bit0:1;
  };
  struct {
    unsigned char byte;
  };
} uint8;


#endif
```

Code for Servo microcontroller

```
#include <htc.h>
#include "delay.h" /* For XTAL_FREQ macros */
#include <stdlib.h>

//A is configured for the tilt servo, B is configured for the roll servo

/* The default servo position after power up (128=centered, 0-255) */
#define SERVO_DEFAULT_POS 128

/* Min and max servo pulse widths in microseconds */
#define SERVO_PERIOD_MIN_US_A 1000L
#define SERVO_PERIOD_MIN_US_B 1085L
#define SERVO_PERIOD_MAX_US_A 1800L
#define SERVO_PERIOD_MAX_US_B 1430L

/* All servos updated once within the following period in microseconds */
#define SERVO_FRAME_US 20000L

//Timer prescaler
#define SERVO_TIMER_DIV 8

//Tick period in us - This is extremely important. This has to be correct or
//everything goes to hell. Unfortunately it gets set to 1, but needs to be
1.6.
//In order to correct for this, we will just aavoid using this variable and
divide
//by 1.6 when needed.
//#define SERVO_TIMER_TICK_US ((4MHZ*SERVO_TIMER_DIV)/(XTAL_FREQ))
#define SERVO_TIMER_TICK_US 1
double servo_tick_period = (double) SERVO_TIMER_TICK_US;

//Min and max servo pulse widths in timer ticks
//#define SERVO_PERIOD_MIN_TICKS (SERVO_PERIOD_MIN_US/SERVO_TIMER_TICK_US)
#define SERVO_PERIOD_MIN_TICKS_A (int)(SERVO_PERIOD_MIN_US_A/1.6)
#define SERVO_PERIOD_MIN_TICKS_B (int)(SERVO_PERIOD_MIN_US_B/1.6)
//#define SERVO_PERIOD_MAX_TICKS (SERVO_PERIOD_MAX_US/SERVO_TIMER_TICK_US)
#define SERVO_PERIOD_MAX_TICKS_A (int)(SERVO_PERIOD_MAX_US_A/1.6)
#define SERVO_PERIOD_MAX_TICKS_B (int)(SERVO_PERIOD_MAX_US_B/1.6)

//Number of timer ticks in the whole servo frame, how many tick lengths fit
into total frame
#define SERVO_FRAME_TICKS (int)(SERVO_FRAME_US/1.6)
double period_total_ticks = (double) SERVO_FRAME_TICKS;

/* The servo output pin */
#define SERVO_PIN_A RB1
#define SERVO_PIN_B RB2

__CONFIG(WDTDIS & HS & UNPROTECT);

//The servo period in ticks, and the pulsewidth in ticks
unsigned int PERIOD= SERVO_FRAME_TICKS;
unsigned int PULSEWIDTH_A;
```

```c
unsigned int PULSEWIDTH_B;

//This is treated as a boolean to switch between emitting a pulse and not
emitting a pulse
unsigned int pulsing_A = 1;
unsigned int pulsing_B = 0;
unsigned int position_A;
unsigned int position_B;

void InitServo(void);
void SetServoPos(unsigned char pos, char servo_ID);
void IncrDecrServoPos(unsigned int dir, char servo_ID);

//unsigned int passvar;
/**
 * Basic main function just for simulation.
 * Set up port pins, initialise the servo PWM timer and enable interrupts.
 */
void main(void) {

      //Set up PORTc as an input
      TRISC = 0xFF;

      //Set up PORTB as an output, then set it low
      TRISB = 0x00;
      PORTB = 0x00;
      while (RC4 != 1)
            ;
      InitServo();

      //INTCON register, not sure what it does
      PEIE = 1; //Peripheral interrupt enable
      GIE = 1; //Global interrupt enable
      unsigned int count = 0;
      for (;;) {
            if (RC4 == 1 && count%15 == 0) {
                  if (count == 15)
                        count = 0;
                  if (RC2 == 1)
                        IncrDecrServoPos(1, 'A');
                  if (RC1 == 1)
                        IncrDecrServoPos(0, 'A');
            }
            if (RC4 == 0 && count%15 == 0) {
                  if (count == 15)
                        count = 0;
                  if (RC2 == 1)
                        IncrDecrServoPos(1, 'B');
                  if (RC1 == 1)
                        IncrDecrServoPos(0, 'B');
            }
            count = count+1;
            //NOP();
      }
}
```

```
/**
 * Initialise the servo PWM timer and set initial servo positions.
 */
void InitServo(void) {
      unsigned char i;

      /* Set up servo timer */
#if SERVO_TIMER_DIV == 1
      T1CON = 0x00;
#elif SERVO_TIMER_DIV == 2
      T1CON = 0x10;
#elif SERVO_TIMER_DIV == 4
      T1CON = 0x20;
#elif SERVO_TIMER_DIV == 8
      T1CON = 0x30;
#else
#error Invalid prescaler value for Timer1
#endif

      SetServoPos(SERVO_DEFAULT_POS, 'A');
      SetServoPos(SERVO_DEFAULT_POS, 'B');
      //Clearing any pending interrupts
      //Enable interrupts
      //Switch the timer on
      TMR1IF = 0;
      TMR1IE = 1;
      TMR1ON = 1;
}

/**
 * Timer interrupt to run low speed servo PWM.
 */
void interrupt isr( void ) {
      static unsigned int total_ticks=0;
      unsigned int low_ticks;
      //TMR1IE - Interrupt Enable (turns interrupt on and off)
      //TMR1IF - Interrupt Flag (goes high when interrupt occurs)
      if(TMR1IE && TMR1IF) {
            TMR1IF = 0;
            TMR1ON = 0;

            if(pulsing_A) {
                  //Set the timer to wait until the pulse is done before
interrupting
                  TMR1H = (unsigned char)((65535-PULSEWIDTH_A) >> 8);
                  TMR1L = (unsigned char)((65535-PULSEWIDTH_A) & 0xFF);
                  TMR1ON = 1;

                  //Start servo pulse and set the pulsing flag to 0 for the
next interrupt
                  //Stop servo B pulse, Start servo A pulse
                  SERVO_PIN_B = 0;
                  SERVO_PIN_A = 1;
                  pulsing_A = 0;
                  pulsing_B = 1;
            } else if(pulsing_B) {
                  /* Finish servo pulse, use up the rest of the frame period
*/
                  TMR1H = (unsigned char)((65535-PULSEWIDTH_B) >> 8);
                  TMR1L = (unsigned char)((65535-PULSEWIDTH_B) & 0xFF);
                  TMR1ON = 1;

                  //Stop servo A pulse, Start servo B pulse
                  SERVO_PIN_A = 0;
                  SERVO_PIN_B = 1;
                  pulsing_B = 0;
            } else {

                  //Set the timer to wait for the rest of the frame before
interrupting again
                  low_ticks = SERVO_FRAME_TICKS - PULSEWIDTH_A -
PULSEWIDTH_B;
                  TMR1H = (unsigned char)((65535-low_ticks) >> 8);
                  TMR1L = (unsigned char)((65535-low_ticks) & 0xFF);
                  TMR1ON = 1;

                  SERVO_PIN_B = 0;
                  //Reset for the next frame
                  total_ticks = 0;
                  pulsing_A = 1;
            }
      }
}

/**
 * Set a servo position.
 * 0 = fully anti-clockwise, 128 = middle, 255 = fully clockwise
 */
void SetServoPos(unsigned char pos, char servo_ID) {
      unsigned int ticks;
      unsigned int min_ticks;
      unsigned int max_ticks;
      if (servo_ID == 'A') {
            position_A = pos;
            min_ticks = SERVO_PERIOD_MIN_TICKS_A;
            max_ticks = SERVO_PERIOD_MAX_TICKS_A;
      } else if (servo_ID == 'B') {
            position_B = pos;
            min_ticks = SERVO_PERIOD_MIN_TICKS_B;
            max_ticks = SERVO_PERIOD_MAX_TICKS_B;
      } else
            return;

      /* Convert the 8 bit position into the pulse width in timer ticks */
      //ticks = (2*pos*(SERVO_PERIOD_MAX_TICKS -
SERVO_PERIOD_MIN_TICKS))/(255*1.6);
      ticks = (int)((max_ticks - min_ticks)*((float)pos/255));

      /* Set the pulse width with interrupts disabled (ISR uses this array)
*/
      di();
      if (servo_ID == 'A')
```

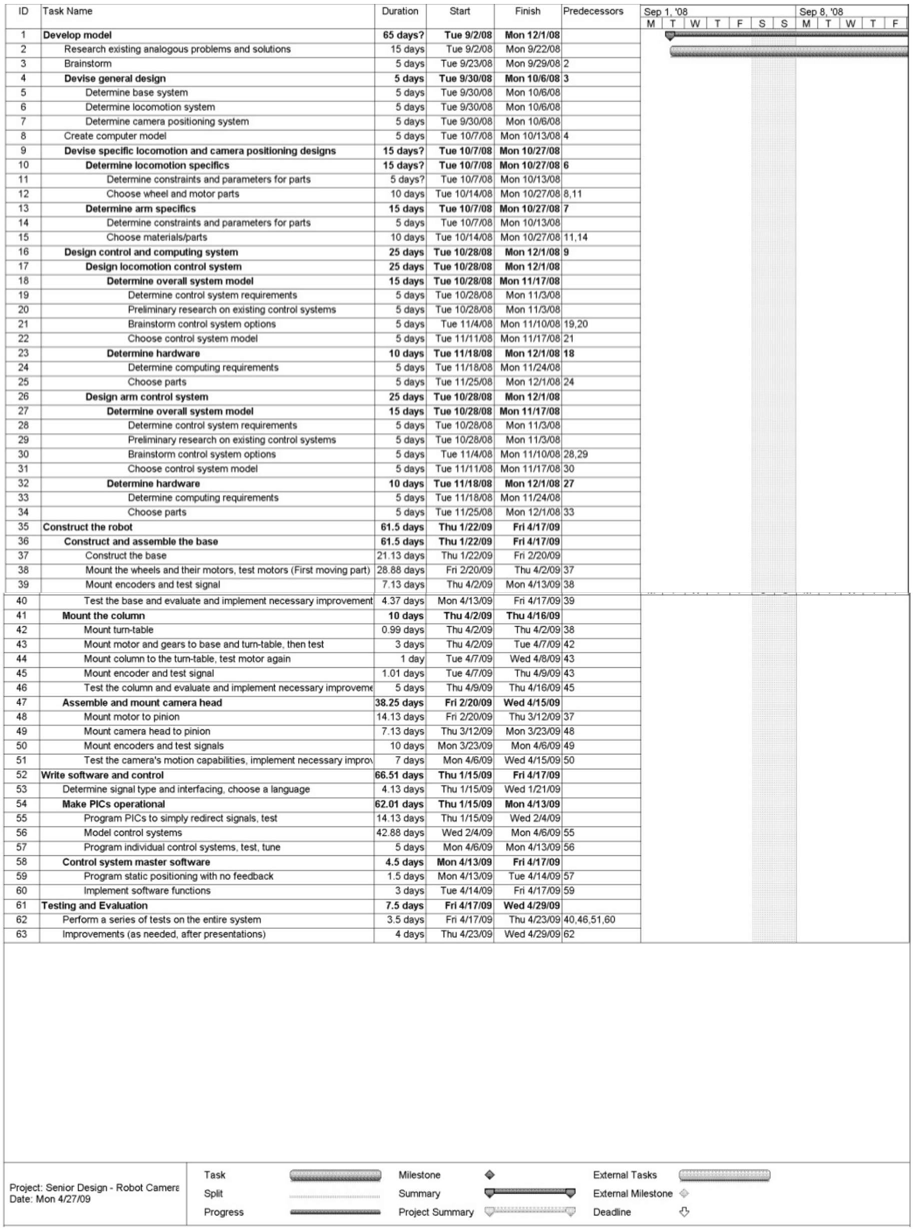
```
            PULSEWIDTH_A = ticks+SERVO_PERIOD_MIN_TICKS_A;
      else if (servo_ID == 'B')
            PULSEWIDTH_B = ticks+SERVO_PERIOD_MIN_TICKS_B;
      ei();

}

void IncrDecrServoPos(unsigned int dir, char servo_ID) {
      if (servo_ID=='A') {
            if (dir == 1 && position_A < 255)
                  position_A = position_A + 1;
            else if (dir == 0 && position_A > 0)
                  position_A = position_A - 1;
            else
                  return;
            SetServoPos(position_A, servo_ID);
      } else if (servo_ID=='B') {
            if (dir == 1 && position_B < 255)
                  position_B = position_B + 1;
            else if (dir == 0 && position_B > 0)
                  position_B = position_B - 1;
            else
                  return;
            SetServoPos(position_B, servo_ID);
      }
      return;
}
```

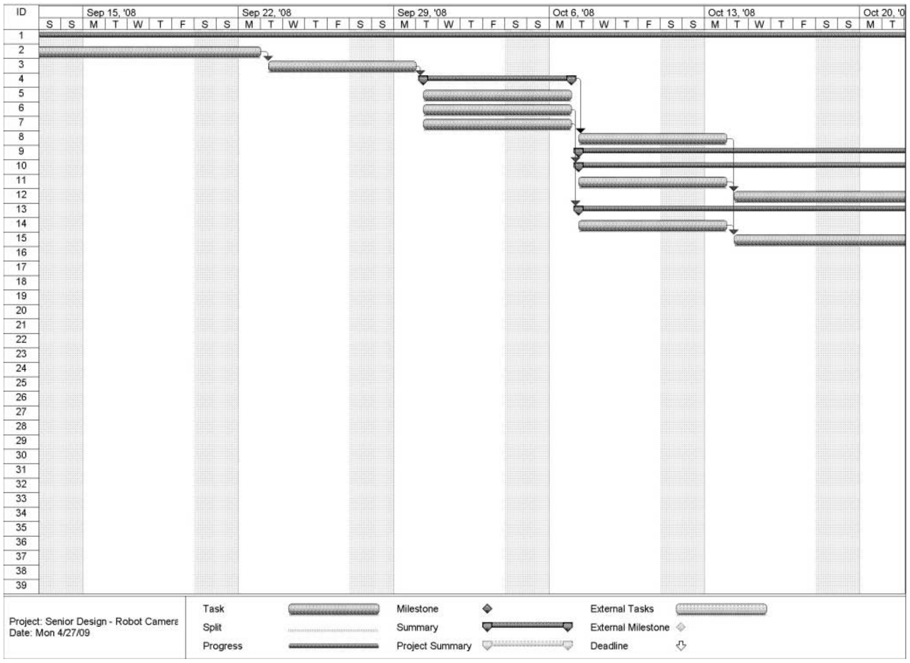
C Schedule

The work breakdown schedule (WBS) and gantt chart were developed in Microsoft Project. The schedule for both the Fall and Spring semesters are included.

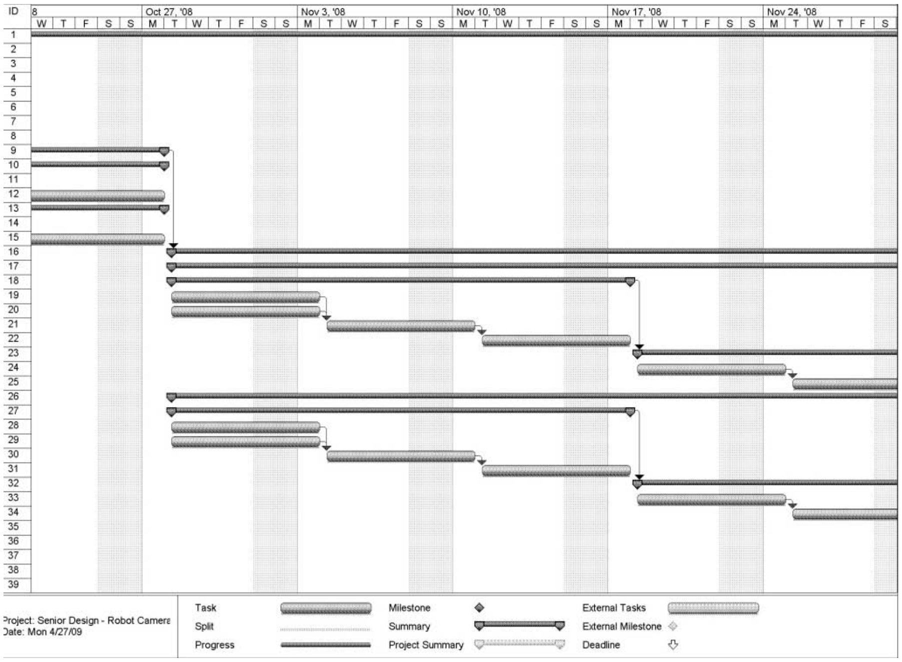


ID	Task Name	Duration	Start	Finish	Predecessors
1	**Develop model**	**65 days?**	**Tue 9/2/08**	**Mon 12/1/08**	
2	Research existing analogous problems and solutions	15 days	Tue 9/2/08	Mon 9/22/08	
3	Brainstorm	5 days	Tue 9/23/08	Mon 9/29/08	2
4	**Devise general design**	**5 days**	**Tue 9/30/08**	**Mon 10/6/08**	**3**
5	Determine base system	5 days	Tue 9/30/08	Mon 10/6/08	
6	Determine locomotion system	5 days	Tue 9/30/08	Mon 10/6/08	
7	Determine camera positioning system	5 days	Tue 9/30/08	Mon 10/6/08	
8	Create computer model	5 days	Tue 10/7/08	Mon 10/13/08	4
9	**Devise specific locomotion and camera positioning designs**	**15 days?**	**Tue 10/7/08**	**Mon 10/27/08**	
10	**Determine locomotion specifics**	**15 days?**	**Tue 10/7/08**	**Mon 10/27/08**	**6**
11	Determine constraints and parameters for parts	5 days?	Tue 10/7/08	Mon 10/13/08	
12	Choose wheel and motor parts	10 days	Tue 10/14/08	Mon 10/27/08	8,11
13	**Determine arm specifics**	**15 days**	**Tue 10/7/08**	**Mon 10/27/08**	**7**
14	Determine constraints and parameters for parts	5 days	Tue 10/7/08	Mon 10/13/08	
15	Choose materials/parts	10 days	Tue 10/14/08	Mon 10/27/08	11,14
16	**Design control and computing system**	**25 days**	**Tue 10/28/08**	**Mon 12/1/08**	**9**
17	**Design locomotion control system**	**25 days**	**Tue 10/28/08**	**Mon 12/1/08**	
18	**Determine overall system model**	**15 days**	**Tue 10/28/08**	**Mon 11/17/08**	
19	Determine control system requirements	5 days	Tue 10/28/08	Mon 11/3/08	
20	Preliminary research on existing control systems	5 days	Tue 10/28/08	Mon 11/3/08	
21	Brainstorm control system options	5 days	Tue 11/4/08	Mon 11/10/08	19,20
22	Choose control system model	5 days	Tue 11/11/08	Mon 11/17/08	21
23	**Determine hardware**	**10 days**	**Tue 11/18/08**	**Mon 12/1/08**	**18**
24	Determine computing requirements	5 days	Tue 11/18/08	Mon 11/24/08	
25	Choose parts	5 days	Tue 11/25/08	Mon 12/1/08	24
26	**Design arm control system**	**25 days**	**Tue 10/28/08**	**Mon 12/1/08**	
27	**Determine overall system model**	**15 days**	**Tue 10/28/08**	**Mon 11/17/08**	
28	Determine control system requirements	5 days	Tue 10/28/08	Mon 11/3/08	
29	Preliminary research on existing control systems	5 days	Tue 10/28/08	Mon 11/3/08	
30	Brainstorm control system options	5 days	Tue 11/4/08	Mon 11/10/08	28,29
31	Choose control system model	5 days	Tue 11/11/08	Mon 11/17/08	30
32	**Determine hardware**	**10 days**	**Tue 11/18/08**	**Mon 12/1/08**	**27**
33	Determine computing requirements	5 days	Tue 11/18/08	Mon 11/24/08	
34	Choose parts	5 days	Tue 11/25/08	Mon 12/1/08	33
35	**Construct the robot**	**61.5 days**	**Thu 1/22/09**	**Fri 4/17/09**	
36	**Construct and assemble the base**	**61.5 days**	**Thu 1/22/09**	**Fri 4/17/09**	
37	Construct the base	21.13 days	Thu 1/22/09	Fri 2/20/09	
38	Mount the wheels and their motors, test motors (First moving part)	28.88 days	Fri 2/20/09	Thu 4/2/09	37
39	Mount encoders and test signal	7.13 days	Thu 4/2/09	Mon 4/13/09	38
40	Test the base and evaluate and implement necessary improvement	4.37 days	Mon 4/13/09	Fri 4/17/09	39
41	**Mount the column**	**10 days**	**Thu 4/2/09**	**Thu 4/16/09**	
42	Mount turn-table	0.99 days	Thu 4/2/09	Thu 4/2/09	38
43	Mount motor and gears to base and turn-table, then test	3 days	Thu 4/2/09	Tue 4/7/09	42
44	Mount column to the turn-table, test motor again	1 day	Tue 4/7/09	Wed 4/8/09	43
45	Mount encoder and test signal	1.01 days	Tue 4/7/09	Thu 4/9/09	43
46	Test the column and evaluate and implement necessary improveme	5 days	Thu 4/9/09	Thu 4/16/09	45
47	**Assemble and mount camera head**	**38.25 days**	**Fri 2/20/09**	**Wed 4/15/09**	
48	Mount motor to pinion	14.13 days	Fri 2/20/09	Thu 3/12/09	37
49	Mount camera head to pinion	7.13 days	Thu 3/12/09	Mon 3/23/09	48
50	Mount encoders and test signals	10 days	Mon 3/23/09	Mon 4/6/09	49
51	Test the camera's motion capabilities, implement necessary improv	7 days	Mon 4/6/09	Wed 4/15/09	50
52	**Write software and control**	**66.51 days**	**Thu 1/15/09**	**Fri 4/17/09**	
53	Determine signal type and interfacing, choose a language	4.13 days	Thu 1/15/09	Wed 1/21/09	
54	**Make PICs operational**	**62.01 days**	**Thu 1/15/09**	**Mon 4/13/09**	
55	Program PICs to simply redirect signals, test	14.13 days	Thu 1/15/09	Wed 2/4/09	
56	Model control systems	42.88 days	Wed 2/4/09	Mon 4/6/09	55
57	Program individual control systems, test, tune	5 days	Mon 4/6/09	Mon 4/13/09	56
58	**Control system master software**	**4.5 days**	**Mon 4/13/09**	**Fri 4/17/09**	
59	Program static positioning with no feedback	1.5 days	Mon 4/13/09	Tue 4/14/09	57
60	Implement software functions	3 days	Tue 4/14/09	Fri 4/17/09	59
61	**Testing and Evaluation**	**7.5 days**	**Fri 4/17/09**	**Wed 4/29/09**	
62	Perform a series of tests on the entire system	3.5 days	Fri 4/17/09	Thu 4/23/09	40,46,51,60
63	Improvements (as needed, after presentations)	4 days	Thu 4/23/09	Wed 4/29/09	62

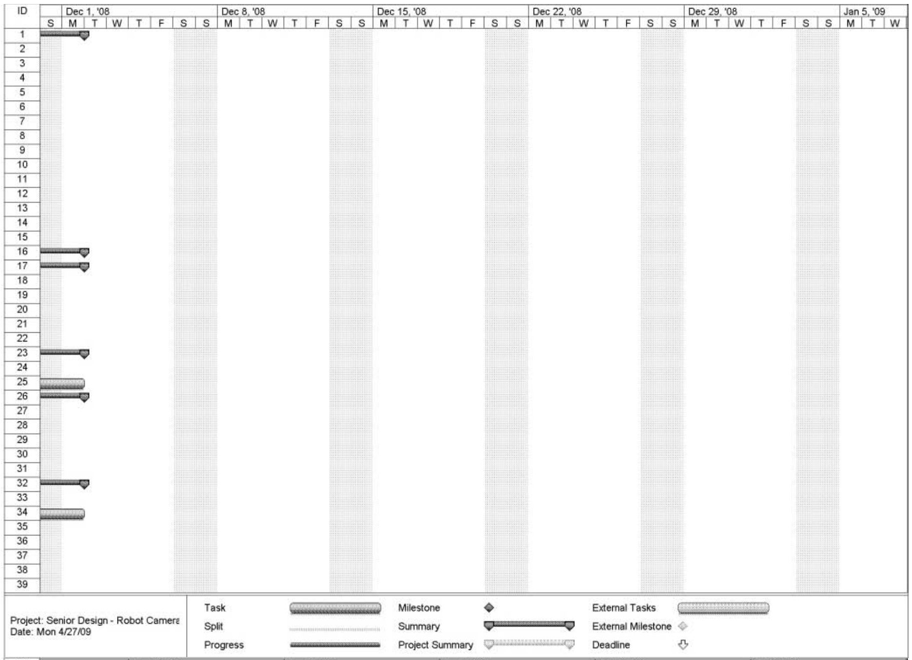
Project: Senior Design - Robot Camera
Date: Mon 4/27/09

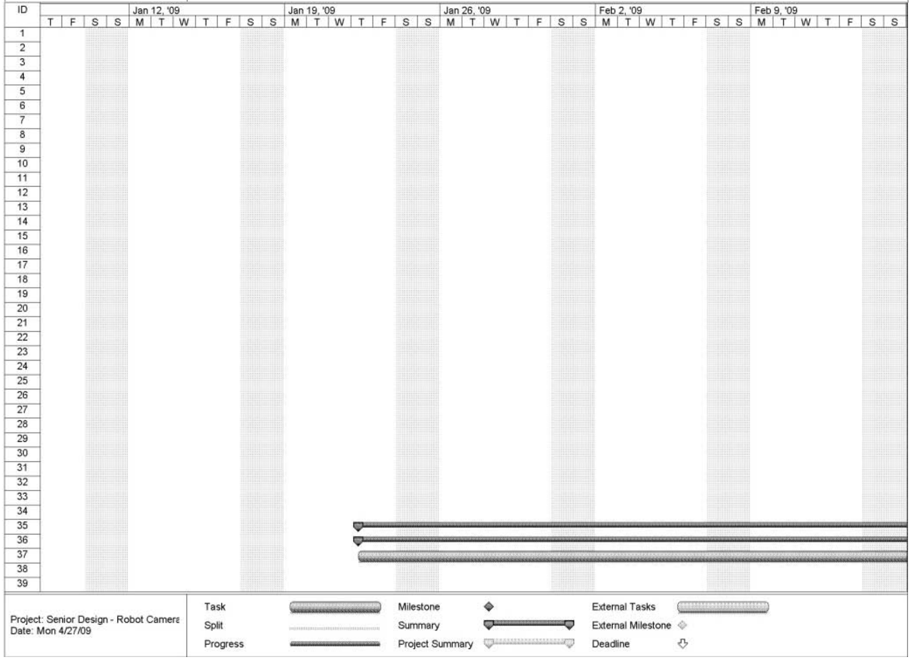


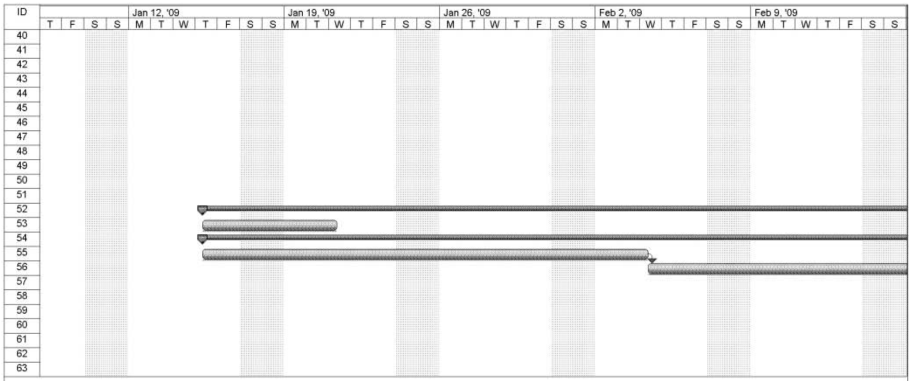
Project: Senior Design - Robot Camera
Date: Mon 4/27/09



Project: Senior Design - Robot Camera
Date: Mon 4/27/09

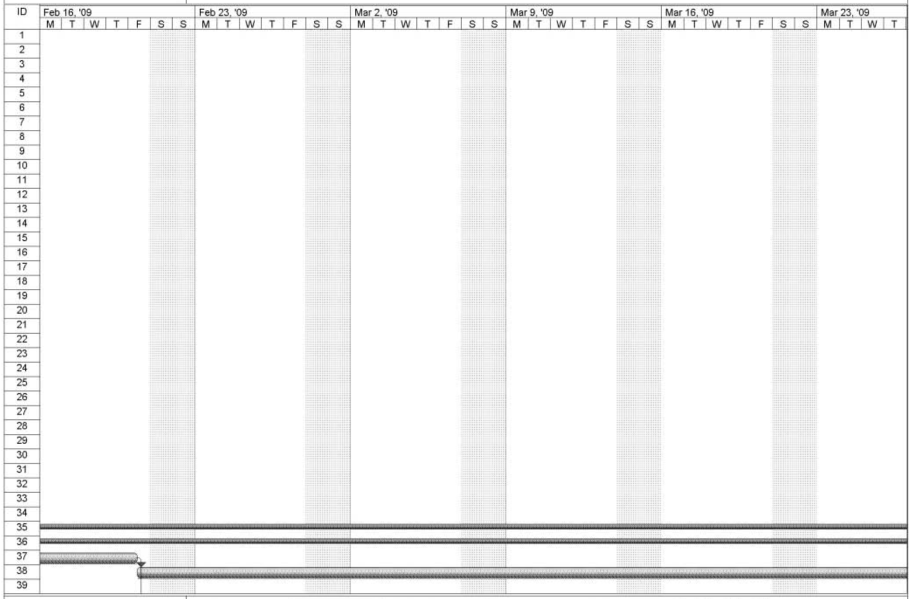

ID
Dec 1, '08
Dec 8, '08
Dec 15, '08
Dec 22, '08
Dec 29, '08
Jan 5, '09
S M T W T F S
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
Project: Senior Design - Robot Camera
Date: Mon 4/27/09
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline


ID
Jan 12, '09
Jan 19, '09
Jan 26, '09
Feb 2, '09
Feb 9, '09
T F S S M T W
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
Project: Senior Design - Robot Camera
Date: Mon 4/27/09
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline



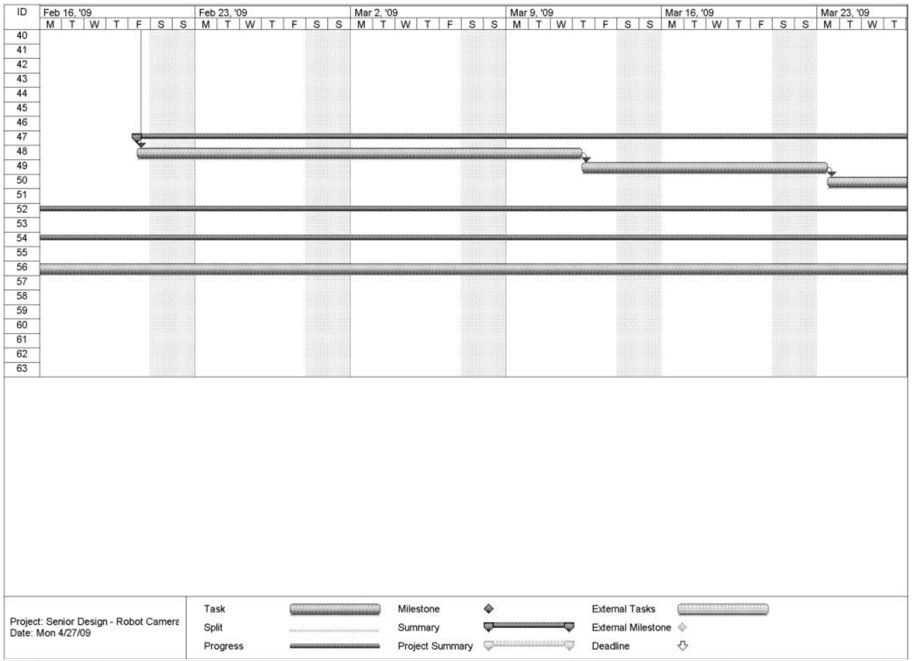
Project: Senior Design - Robot Camera
Date: Mon 4/27/09

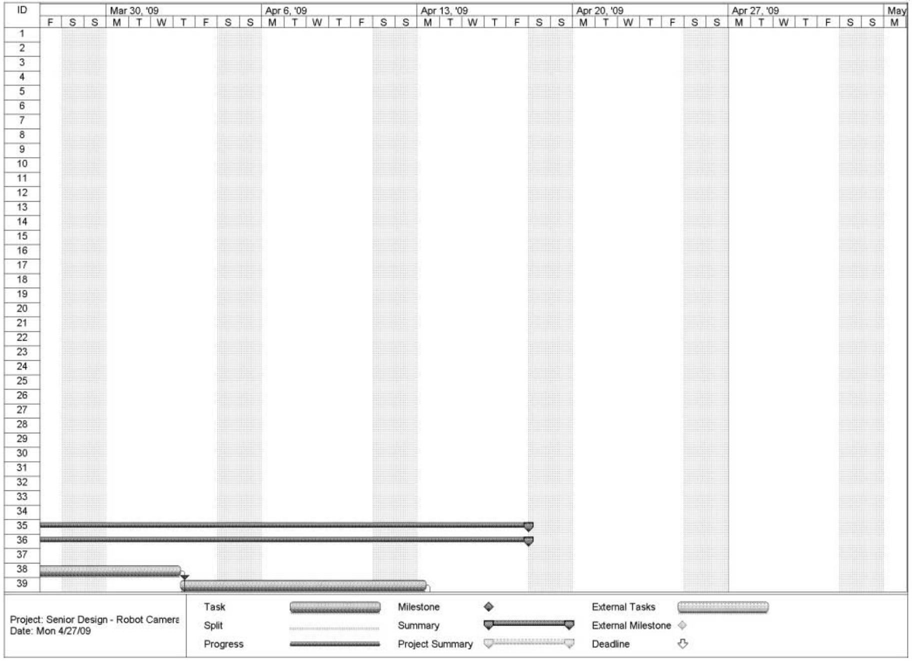
Task ▨ Milestone ◆ External Tasks ▨
Split External Milestone ◇
Progress Summary External Milestone ◇
Project Summary Deadline ⇩

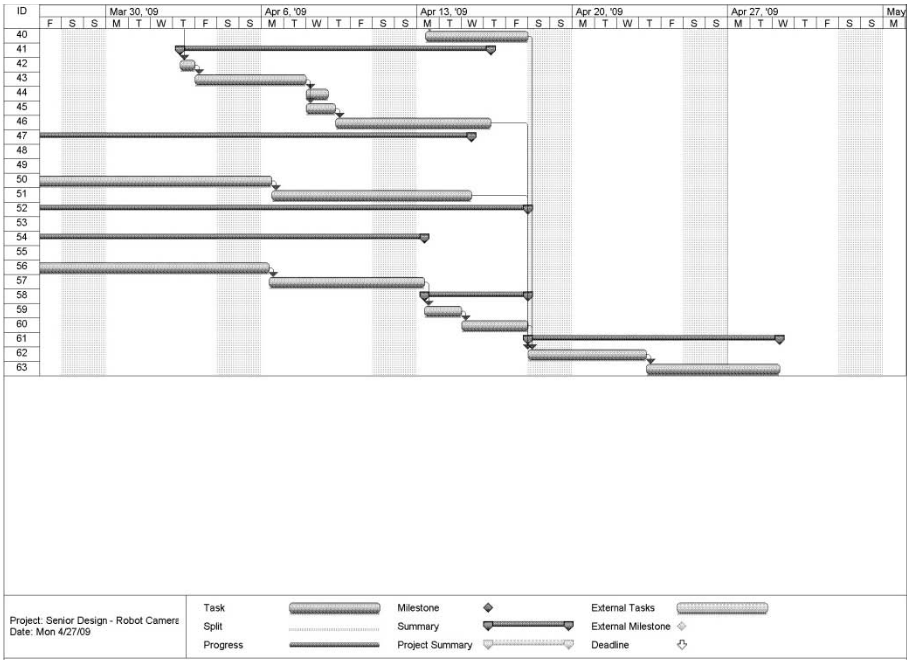


Project: Senior Design - Robot Camera
Date: Mon 4/27/09

Task ▨ Milestone ◆ External Tasks ▨
Split Summary External Milestone ◇
Progress Project Summary Deadline ⇩


ID
Feb 16, '09
Feb 23, '09
Mar 2, '09
Mar 9, '09
Mar 16, '09
Mar 23, '09
M T W T F S S
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
Project: Senior Design - Robot Camera
Date: Mon 4/27/09
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline


ID
Mar 30, '09
Apr 6, '09
Apr 13, '09
Apr 20, '09
Apr 27, '09
May
F S S M T W T
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
Project: Senior Design - Robot Camera
Date: Mon 4/27/09
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline


ID
Mar 30, '09
Apr 6, '09
Apr 13, '09
Apr 20, '09
Apr 27, '09
May
F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
Project: Senior Design - Robot Camera
Date: Mon 4/27/09
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline

D Budget

The budget alotted includes the money provided by the department, $1200.00, and the IEEE grant money, $1500.00. When purchases included many small parts, such as electronics and nuts and bolts, the total purchase amount is a combination of all the items.

Income

Date	**Sponsor**	**Description**	**Budgeted Amount**	**Actual Amount**
9/1/2004	Engr Dept	Seed Money	$1,200.00	$1,200.00
11/9/2004	IEEE	Grant Money	$1,500.00	$1,500.00
Total Income				$2,700.00

Expenses

						Status (Check one)		
Date	**Vendor**	**PO #**	**Item Description**	**Status (Planned/ Pending/ Cleared)**	**Amount**	**Dept PO**	**PCARD**	**Reimburse ment**
12/3/2008	Servo City	DDT560H	Direct Drive Cradle Tilt	Cleared	$49.99		X	
12/3/2008	Servo City	GDP785A-BM	Gear Drive Pan System	Cleared	$60.00		X	
12/3/2008	Servo City	33785S	HS-785HB Servo (Universal Connector)	Cleared	$39.99		X	
12/3/2008	Servo City	32985S	HS-985MG Servo (Universal Connector)	Cleared	$72.99		X	
12/3/2008	Servo City	5445	180 Rotation Modfication	Cleared	$10.00		X	
12/3/2008	Servo City	902MSD	Dual Manual Servo Controller	Cleared	$49.99		X	
12/3/2008	Servo City	418-TR1506-12	6VDC, 1.5amp Linear Power Supply (2.5mm plug)	Cleared	$24.99		X	
12/3/2008	Servo City	180SS	180 Servo Stretcher	Cleared	$19.99		X	
12/15/2008	RobotEQ	AX1500	Cost Optimized, 2 x 30A Brushed DC Motor	Cleared	$288.00		X	
12/15/2008	SparkFun Electronics	COM-00226	PIC 40 Pin 20MHz 8K 8A/D-16F877A+shipping	Cleared	$66.05		X	
12/15/2008	SparkFun Electronics	DEV-07830	Ethernet Web PIC+shipping	Cleared	$55.35		X	
12/15/2008	Monsterscooterparts	MOI-385	36V 1000W Direct Drive Electric Motor & Rear Wheel	Cleared	$384.32		X	
12/15/2008	Servo City	6031K19	12" Ball Bearing Turntable	Cleared	$25.90		X	
1/15/2009	Robotmarketplace	SP-40FB19x.875	1/2 pitch Type B Sprocket-19 teeth, 7/8' bore+shipping	Cleared	$23.93		X	
1/15/2009	Robotmarketplace	CHA-40-HALF	#40 Roller Chain-10ft length+shipping	Cleared	$13.40		X	
1/15/2009	Robotmarketplace	CHA-40-10	4, #40 Roller Chain offset link+shipping	Cleared	$36.50		X	
1/15/2009	Bearing Agencies	BP5100850-20	DeWalt Motor/Gearbox+shipping	Cleared	$56.99		X	
1/15/2009	Mouser Electonics	AEAT-6010-A06	4 10 bit ABS Magnetic Encoder	Cleared	$111.72		X	
1/22/2009	Triple S Steel	--	1'x1' 11 GA. X 24' Square Tubing	Cleared	$26.40			X
1/22/2009	The Home Depot	99167212838	18MMBirch	Cleared	$26.56			X
2/25/2009	Intertex Electronics	--	Micellaneous Electronics	Cleared	$8.06			X
2/26/2009	McMaster Carr	1710K4	3/4"-5 Platform Nut+shipping	Cleared	$129.25		X	

2/26/2009	McMaster Carr	98940A361	3/4"-5 Acme 6' Rod+shipping	Cleared	$113.90		X	
2/26/2009	Digi-Key	--	Micellaneous Electronics	Cleared	$26.71			X
3/24/2009	Autozone	298374	24MD-DL Duralast Marine Deep Cycle Battery	Cleared	$62.99			X
3/24/2009	Autozone	298374	Core Charge	Cleared	$12.00			X
3/24/2009	Autozone	570475	2 Duralast 4GA 19" Switch Starter Battery Cables	Cleared	$8.98			X
3/24/2009	Walmart	3399100211	FSD Dist. Block	Cleared	$9.82			X
3/24/2009	Purvis Industries	203PP030867	Fafnir	Cleared	$22.70			X
3/25/2009	Grainger	4PU79	2 Bearing, 3/4 In Bore	Cleared	$117.28			X
3/25/2009	The Home Depot	--	Tower Construction Misc. Parts	Cleared	$20.00			X
3/25/2009	Lowe's	--	Tower Construction Misc. Parts	Cleared	$56.50			X
3/29/2009	Robotmarketplace	SH-X-1400-RL	18 7/8" diameter keyed shaft+shipping	Cleared	$46.59			X
3/29/2009	Robotmarketplace	KEY-407-0300	3/16" key stock 12" long (shipping included above)	Cleared	$2.87			X
3/29/2009	Lowe's	--	Tower Construction Misc. Parts	Cleared	$46.41			X
3/29/2009	Home Depot	--	Tower Construction Misc. Parts	Cleared	$28.18			X
4/1/2009	Intertex Electronics	IHD12-6.8	1x12V 6.8A Linear Power Supply IHD12-6.8	Cleared	$98.03	X		X
4/1/2009	Robotmarketplace	--	Gearing	Cleared	$65.21	X		X
4/1/2009	The Home Depot	--	Tower Construction Misc. Parts	Cleared	$17.89			X
4/1/2009	Sail & Ski Center	27501	CGJ General Safety Switch w/lanyard	Cleared	$20.93			X
4/1/2009	Intertex Electronics	--	Micellaneous Electronics	Cleared	$53.92	X		X
4/1/2009	Monsterscooterparts	--	24V 450W Electric Motor	Cleared	$83.66	X		X
4/1/2009	RoboteQ	AX500 SC	AX500 SC	Cleared	$158.00	X		X
4/1/2009	Digi-Key	TLE 5205-2IN-ND	IC H-Bridge 5A DC Motor TO220-7	Cleared	$7.57			X
4/1/2009	Donated by Andy	--	Misc Used Electronic Parts	Cleared	$1.00			
4/1/2009	Borrowed from Andy	--	Playstation Controller	Cleared	$14.99			
4/1/2009	Donated by TU	--	Scrap Metal	Cleared	$0.00			
4/1/2009	TU Engineering Dept	--	330 ohm resistor	Cleared	$0.20			
4/1/2009	TU Engineering Dept	--	1.5k ohm resistor	Cleared	$0.10			
4/1/2009	TU Engineering Dept	--	1 uF capacitor	Cleared	$0.40			
4/1/2009	TU Engineering Dept	--	16 pin socket	Cleared	$0.35			
4/1/2009	TU Engineering Dept	--	22 ga Wire (ft)	Cleared	$2.00			
4/1/2009	TU Engineering Dept	--	DB9 male connector	Cleared	$0.70			
4/27/2009	Ginny's	--	Final Report	Pending	$ 26.28			

Total Expenses		$2,680.24
Donations		$15.99
Budget Remaining	$35.75	

E Bill of materials & list of vendors

The bill of materials (BOM) serves as an itemized list of all the materials required to fabricate the design. The vendor from which the item was purchased is in the left most column; the contact information for all vendors is found following the BOM.

Vendor	PO #	Item Description	Qty.	Cost		Subtotal
Autozone	298374	24MD-DL Duralast Marine Deep Cycle Battery	1	$74.99	per piece	$74.99
Autozone	570475	Duralast 4GA 19" Switch Starter Battery Cables	2	$4.49	per piece	$8.98
Bearing Agencies	BP5100850-20	DeWalt Motor/Gearbox+shipping	1	$56.99	per piece	$56.99
Donated by Andy	--	8 pin connector	1	$0.05	per piece	$0.05
Donated by Andy	--	4 pin connector	1	$0.05	per piece	$0.05
Donated by Andy	--	3 pin connector	2	$0.05	per piece	$0.10
Donated by Andy	--	8 gauge wire (ft)	5	$0.45	per piece	$2.25
Donated by Andy	--	Playstation Controller	1	$14.99	per piece	$14.99
Grainger	4PU79	Bearing, 3/4 In Bore	2	$58.64	per piece	$117.28
Intertex Electronics	ST-40	40 pin IC socket	3	$0.75	per piece	$2.25
Intertex Electronics	CY20	20 MHz crystal	3	$2.95	per piece	$8.85
Intertex Electronics	7805T	5V Voltage Regulator	2	$1.75	per piece	$3.50
Intertex Electronics	--	100 uF capacitor	2	$0.25	per piece	$0.50
Intertex Electronics	--	10uF capacitor	2	$0.20	per piece	$0.40
Intertex Electronics	MAX232N	RS232/TTL Level Shifter	1	$1.35	per piece	$1.35
Intertex Electronics	--	22 pF capacitor	6	$0.25	per piece	$1.50
Intertex Electronics	1N4007	1N4007 diode	2	$0.45	per piece	$0.90
Intertex Electronics	--	9 conductor ribbon cable (ft)	15	$0.15	per piece	$2.25
Lowes	--	1/2"x72" Black Steel Pipe	2	$12.17	per piece	$24.34
Lowes	--	1/2" Floor Flange	4	$3.27	per piece	$13.08
McMaster Carr	1710K4	3/4"-5 Platform Nut+shipping	1	$129.25	per piece	$129.25
McMaster Carr	98940A361	3/4"-5 Acme 6' Rod+shipping	1	$113.90	per piece	$113.90
Monsterscooterparts	MOI-385	36V 1000W Direct Drive Electric Motor & Rear Wheel	2	$192.16	per piece	$384.32
Monsterscooterparts	--	24V 450W Electric Motor	1	$83.66	per piece	$83.66
RoboteQ	AX500 SC	AX500 SC Controller	1	$158.00	per piece	$158.00
RobotEQ	AX1500	Cost Optimized, 2 x 30A Brushed DC Motor	1	$288.00	per piece	$288.00
Robotmarketplace	SP-40FB19x.875	1/2 pitch Type B Sprocket-19 teeth, 7/8' bore+shipping	1	$23.93	per piece	$23.93
Robotmarketplace	SH-X-1400-RL	18 7/8" diameter keyed shaft+shipping	1	$46.59	per piece	$0.00
Robotmarketplace	KEY-407-0300	3/16" key stock 12" long (shipping included above)	1	$2.87	per piece	$0.00
Robotmarketplace	CHA-40-HALF	#40 Roller Chain-10ft length+shipping	1	$13.40	per 10ft length	$13.40
Robotmarketplace	CHA-40-10	#40 Roller Chain offset link+shipping	1	$9.13	per piece	$9.13
Robotmarketplace	CHA-25-10	#25 Roller Chain-10ft length+shipping	1	$24.75	per 10ft length	$24.75
Robotmarketplace	CHA-25-CONN	Roller Chain connecting link	2	$1.00	per piece	$2.00

Vendor	PO #	Item Description	Qty.	Cost		Subtotal
Robotmarketplace	SP-25FB	20x.75 1/4 pitch Type B Sprocket-20 teeth, 3/4 inch bore	1	$24.38	per piece	$24.38
Sail & Ski Center	27501	CGJ General Safety Switch w/lanyard	1	$20.93	per piece	$20.93
Servo City	6031K19	12" Ball Bearing Turntable	1	$25.90	per piece	$25.90
Servo City	DDT560H	Direct Drive Cradle Tilt	1	$49.99	per piece	$49.99
Servo City	GDP785A-BM	Gear Drive Pan System	1	$60.00	per piece	$60.00
Servo City	33785S	HS-785HB Servo (Universal Connector)	1	$39.99	per piece	$39.99
Servo City	32985S	HS-985MG Servo (Universal Connector)	1	$72.99	per piece	$72.99
Servo City	5445	180 Rotation Modfication	1	$10.00	per piece	$10.00
Servo City	902MSD	Dual Manual Servo Controller	1	$49.99	per piece	$49.99
Servo City	418-TR1506-12	6VDC, 1.5amp Linear Power Supply (2.5mm plug)	1	$24.99	per piece	$24.99
Servo City	180SS	180 Servo Stretcher	1	$19.99	per piece	$19.99
Sparkfun Electronics	PIC 16F877A	Microcontroller	3	$8.95	per piece	$26.85
The Home Depot	--	1/4"-20x1" Flat Head Slotted Acero Inoxidable	8	$0.98	pkg 2	$7.84
The Home Depot	--	#6-32x1/2" Round Head Slotted	8	$0.36	per piece	$2.88
The Home Depot	--	4x8 18MMBirch	2	$13.28	per piece	$26.56
The Home Depot	--	1/4"- 20x2" Hex bolts	2	$4.28	pkg 25	$8.56
The Home Depot	--	1/4" Hex Nut/Coarse Thread	1	$5.77	pkg 100	$5.77
The Home Depot	--	6-32x1-1/4" Combo Round w/Nut	1	$1.18	pkg 10	$1.18
The Home Depot	--	Angle Gauge-3'	1	$8.97	per piece	$8.97
Triple S Steel	--	1'x1' 11 GA. X 24' Square Tubing	1	$26.40	per 20ft length	$26.40
TU Engineering Dept	--	330 ohm resistor	2	$0.10	per piece	$0.20
TU Engineering Dept	--	1.5k ohm resistor	1	$0.10	per piece	$0.10
TU Engineering Dept	--	1 uF capacitor	5	$0.08	per piece	$0.40
TU Engineering Dept	--	16 pin socket	1	$0.35	per piece	$0.35
TU Engineering Dept	--	22 ga Wire (ft)	20	$0.10	per piece	$2.00
TU Engineering Dept	--	DB9 male connector	2	$0.35	per piece	$0.70
TU Engineering Dept	--	1/8"x 2 1/2"x 8"Steel Plate-Scrap Metal	2	$0.00	per piece	$0.00
Walmart	3399100211	FSD Dist. Block	1	$9.82	per piece	$9.82

Autozone

2223 Blanco Rd
San Antonio TX 78212
(210) 737-1255
http://www.autozone.com

Bearing Agencies Inc.

277 7th St
San Francisco, CA 94103
(415) 621-8363
Fax: (415) 621-7239

Digi-Key Corporation

701 Brooks Avenue South
Thief River Falls, MN 56701 USA
(800) 344-4539
Fax: (218) 681-3380
http://www.digikey.com

Grainger

5011 Rittman Rd.
San Antonio, TX 78218-4638
(210) 654-4020
http://www.grainger.com

Intertex Electronics

1200 W.Hildebrand
San Antonio,TX 78201
(800) 820-3908
Fax: (210) 820-3344
http://www.intertexelectronics.com

Lowe's

1470 Austin Highway
San Antonio, TX 78209
(210) 828-6011
Fax: (210) 828-3866
http://www.lowes.com

McMaster Carr

200 New Canton Way
Robbinsville, NJ 08691-2343
(609) 689-3000
Fax: (609) 259-3575
http://www.mcmaster.com

Monsterscooterparts

26262 Three North Road Unit 6
Mechanicsville, MD 20659
(800) 798-0325
http://www.monsterscooterparts.com

The Home Depot
435 Sunset Rd West
San Antonio, TX 78209
(210) 824-9677
http://www.homedepot.com

Purvis Industries
611 N. WW White Road
(210) 299-1010
FAX (210) 226-1790
http://purvisindustries.com

RobotEQ
8426 E. Shea Blvd.
Scottsdale, AZ 85260
(602) 617-3931
http://www.roboteq.com/

Robot Market Place
5129A 53rd Avenue East
Bradenton, FL 34203
1-877-ROBOT 99
Fax#(941) 753-5113
http://www.robotmarketplace.com

Mouser Electronics
1000 North Main Street
Mansfield, TX 76063
(800) 346-6873
Fax: (817) 804-3899
http://www.mouser.com

Servo City
620 Industrial Park
Winfield, KS 67156
(620) 221-0123
Fax : (620) 221-0858
www.servocity.com

Triple S Steel
2042 Thompson Pl
San Antonio, TX 78226
(210) 431-0088
Fax: (210)431-0701
http://www.sss-steel.com

Trinity University
1 Trinity Place
San Antonio, TX 78212
(210) 999-7511
Fax#:(210) 999-8037
www.trinity.edu

Sail & Ski Center

141 Balcones North
San Antonio, TX 78201 USA
(210) 734-8199
Fax: (210) 734-8130
http://www.sailandski.com

SparkFun Electronics

6175 Longbow Dr. Suite 200
Boulder, CO 80301
(303) 284-0979
Fax: (303) 443-0048
http://www.sparkfun.com

Walmart

1430 Austin Highway
San Antonio (E), TX 78209
(210) 637-1700
http://www.walmart.com