ALiCE II:
Assembly of an Autonomous Line-Crawling Robot Designed for Manitoba Hydro

Maciej Borkowski
Christopher Henry
Dan Lockery
Peter Schilling
James F. Peters, Supervisor
{*maciey,dlockery,chenry,jfpeters*}*@ee.umanitoba.ca*
University of Manitoba
Computational Intelligence Laboratory
ENGR E3-576 Engineering & Information Technology Complex
75A Chancellor's Circle
Winnipeg, Manitoba R3T 5V6
Tel.: 204.474.9603
Fax.: 204.261.4639

Submitted 10 August 2005, revised 4 April 2006

UM CI Laboratory Technical Report
Number TR-2006-010
April 8, 2006

University of Manitoba ALiCE II

Computational Intelligence Laboratory
URL: http://www.ee.umanitoba.ca/research/cilab.html

Project no. 2006-012

ALiCE II:
Assembly of an Autonomous Line-Crawling Robot Designed for Manitoba Hydro

Maciej Borkowski
Christopher Henry
Dan Lockery
Peter Schilling
James F. Peters, Supervisor
{maciey,dlockery,chenry,jfpeters}@ee.umanitoba.ca
University of Manitoba
Computational Intelligence Laboratory
ENGR E3-576 Engineering & Information Technology Complex
75A Chancellor's Circle
Winnipeg, Manitoba R3T 5V6
Tel.: 204.474.9603
Fax.: 204.261.4639
Submitted 10 August 2005, revised 4 April 2006

CI Laboratory TR-2006-010

April 8, 2006

Abstract

The is the third in a series of research reports that were begun in 2005, and are now being completed in preparation for the completion of the first phase of the ALiCE II project. This report focuses on assembly of the line-crawling robot affectionately named ALiCE II. ALiCE II is an acronym for Autonomous Line-Crawling Equipment II, a second generation version of a new family of autonomous line-crawling robotic devices using swarm intelligence system engineering design principles introduced during the past 3 years. ALiCE II represents the combined efforts of Maciej Borkowski, Dan Lockery, Christopher Henry (alpha order) with some help from Peter Schilling during the summer of 2005. The main architect of ALiCE II has been Dan Lockery. ALiCE I was a single line-crawling robot designed by Vitaliy Degetyarov in 1999 as part of his M.Sc. project, which was also funded by Manitoba Hydro. ALiCE II is the focus of Dan Lockery's M.Sc. research project.

This research work has been funded by Manitoba Hydro.

Contents

List of Figures

1 Introduction

This report focuses on what has been done with the line crawler assembly over the previous few weeks. This report covers the final assembly of Alice II and our preparations for launching the first bot on the line and gathering some still images to discover how all of the subsystems will work together. We experienced a few more interesting problems and have since solved most of them. Unfortunately we are still waiting on a few remaining parts that have yet to arrive. Likely by the time the notes are finished they may have arrived and in which case, a section will be included in the discussion for what happens next that covers how they will be used.

2 Final Assembly of Alice II

Continuing from the previous report, a number of items have been completed. First, the PIC control software has been extended to cover the protocol that we decided upon for controlling all of the actuated joints and locomotion control. Next, the contact switch sensor interfaces were re-tooled and have been temporarily mounted ready for the experimental phase. In addition to that, wiring holes have been made for the USB camera as well as the two servos that are providing pan and tilt. The servos were all positioned appropriately to provide the proper amount of motion with using the smallest amount of time for each control signal. The new dc motor was mounted on the mounting plate and has already been experimented with to verify that it can manage to operate the power train. We have been working on adapting the code that Maciek provided for operating the camera. Chris has been streamlining the code since there was support for streaming video and we are mainly interested in capturing still images for now. Peter has been continuing work on constructing the second bot. We are still waiting for a few remaining parts on order including some servo extension wires and some proximity sensors but they are expected to arrive shortly.

2.1 PIC Control Software

The first thing that I will be discussing are the changes made to the PIC control software. The new protocol that we are using is designed to be set for one-way communication (from the TS-5500 to the PIC). The reason why we decided to keep it this way was for simplicity and ease if implementation as well as reducing the amount of communication from one controller to the other. We are using the same specifications for the serial link, 9600bps, 8-N-1 standard RS232 protocol. We decided to use a single byte protocol to keep the amount of serial traffic to a minimum and maintain the simplicity of our design.

The first three bits of each byte sent serially from the TS-5500 to the PIC contains the device selection bits. Since we have three bits, there is a possible range of up to 8 devices at present. The order of selection is seen in table 1 below.

The final five bits of each serial byte issued to the PIC provides a command to go along with the respective device selected. The commands vary based on the device selected with each having its own relevant commands to provide the actions that are needed for the line crawler. The dc motor has a very simple set of commands, there are four possibilities, turn the motor clockwise, counter clockwise, apply the brakes, and halt (or enter quiescent state). These four commands round out the dc motor control scheme and provide us with enough control to manage Alice II's locomotion. The grip servos are equally simple to operate since they have initially been limited as to the number of positions that they are capable of achieving. Initially, we are interested in opening and closing each of the upper and lower parts of the grip. As a result, there are no partially open positions at present, only open or closed. Finally, operating the camera has a slightly different approach with step movements allowed in varying increments. The range is from +16 steps to -16 steps with every increment in between possible. Each of the steps in question correspond to the 8.08μS increments discussed in the previous set of notes when looking at programming variable delays. This translates to about

Table 1: Device Selection for PIC Control

Binary	*Decimal*	*Description*
000	0	DC Motor
001	1	Servo #1 selected (Upper grip servo)
010	2	Servo #2 selected (Lower grip servo)
011	3	Servo #3 selected (Pan servo for camera)
100	4	Servo #4 selected (Tilt servo for camera)
101	5	Additional device space
110	6	Additional device space
111	7	Additional device space

a 2 degree movement by each servo motor. The tables below show exactly what the upper 5 bits of the serially communicated byte will do for us dependent upon the device requested.

Table 2: DC Motor Commands

Binary	*Decimal*	*Description*
00000	0	Stop (quiescent state)
00001	1	Brake
00010	2	Turn clockwise
00011	3	Turn counter clockwise

Table 3: Grip Servo Commands

Binary	*Decimal*	*Description*
00000	0	Open Grip
00001	1	Close Grip
00010	2	Reserved for future commands
⋮	⋮	Reserved for future commands
11111	31	Reserved for future commands

This covers the protocol and how each of the different devices will now be controlled via the PIC from serial bytes presented to it. There is room for expansion should additional devices be required to complete any of the tasks that we are interested in performing. Also, there is the possibility of additional commands for the grip servos should we decide that new positioning is needed to achieve certain tasks. This completes the discussion for the changes and additions made to the PIC control software.

Table 4: Camera Servo Control Commands

Binary	*Decimal*	*Description*
10000	16	Move servo -16 steps
10001	17	Move servo -15 steps
⋮	⋮	⋮
11110	30	Move servo -2 steps
11111	31	Move servo -1 step
00000	0	Move servo +1 step
00001	1	Move servo +2 steps
00010	2	Move servo +3 steps
⋮	⋮	⋮
01111	15	Move servo +16 steps

2.2 Improved Contact Switch Interfaces

One of the problems encountered with the initial version of the line crawler grip was that the contact switches were not as well crafted as originally hoped. There were a couple of problems, the first one being that they did not provide a very sensitive interface when obstacles were encountered. The other difficulty faced was that they had a tendency to hang below the tracking of the wheels when the line crawler hit an incline or decline in the original sky wire experiments. With these ideas in mind, Peter put together a new type of interface for the contact switches that addresses these issues.

Both the upper and lower grips required new contact switch interfaces since they were somewhat insensitive to contacts with obstacles in the preliminary trials. To reduce the potential damage to the robot, the drive train and the payload, we decided that it was more important to develop a more sensitive interface to contact the obstacles with. The upper grip interfaces were tackled first and the problem areas that were addressed helped to dictate the new shape and style of triggering used. We were most concerned with keeping the interfaces off of the sky wire during the travels up and down the inclines. As a result, the new shape is offset from the previous location and lifted up enough to avoid contact with the line including moderate inclines/declines. Several images are included to demonstrate how the new interfaces look when they are at rest. To gain a better idea of their operation, a few extra images were included for alternate profiles.

The lower grip contact switch interface was also modified but this time it was mainly due to the lack of sensitivity found in the preliminary version. We opted for a slight change in shape which encouraged triggering at much less input pressure and the position is slightly more favourable for contacting the vibration dampers which is largely the role of the lower contact switches. The previous images give a view of how they are attached as well as how they will trigger the contact switches. Once we have begun the experimental work, supporting images will be provided detailing how each obstacle is meant to trigger its respective contact switch. With these interfaces in place, we are all set to exercise the contact switches once the remainder of the work is finished.

2.3 Wiring the Camera and Servos

In order to get the hardware that resides under the platform wired to the TS-5500, we needed to create some sort of hole or passage for the cables. Since the connector for the camera is USB, we needed a hole with a diameter of approximately 5/16 of an inch. To drill the hole, we used a spade bit that is meant for

Figure 1: Upper Grip Contact Switch Interface

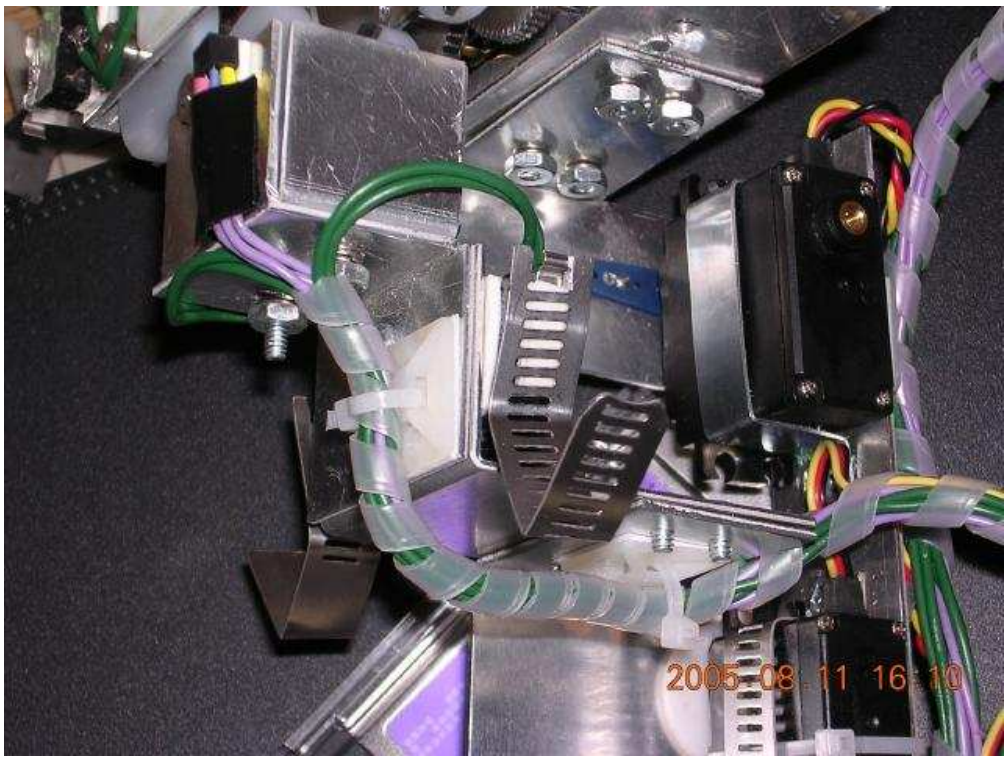

Figure 2: Lower Grip Contact Switch Interface

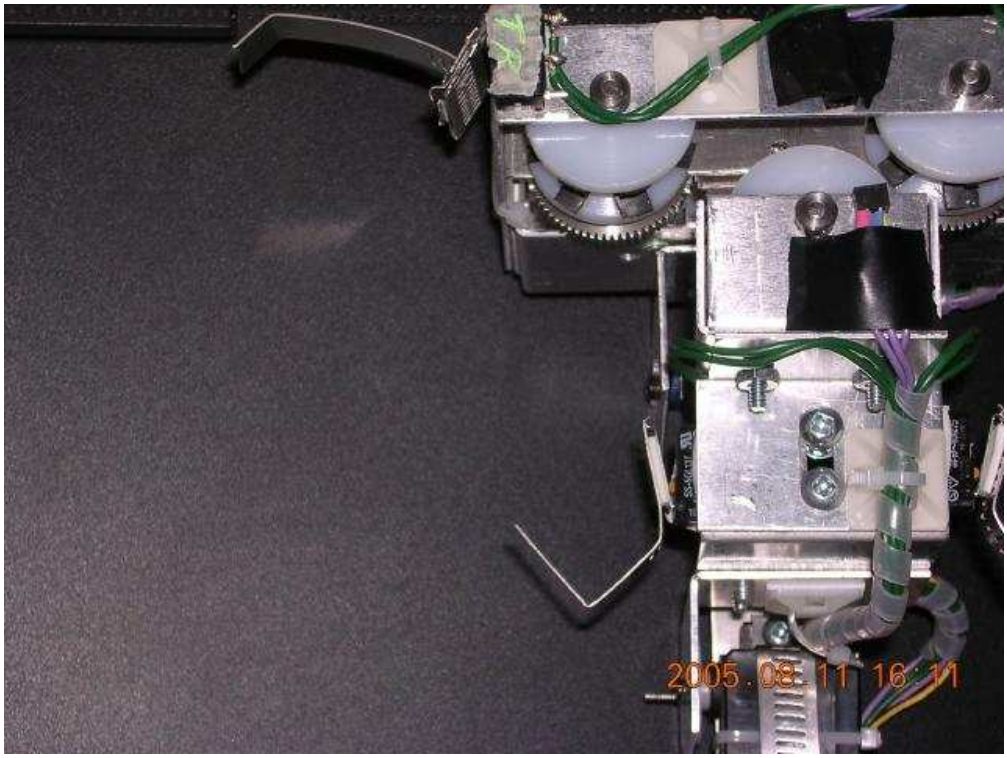

Figure 3: Front Profile View of Upper and Lower Contact Switches

wood. Fortunately for us, the platform material is Aluminum (1100 series), so it is a reasonably malleable substance and easily drilled, even for larger hole sizes.

We decided to opt for a 3/4 inch hole, allowing some extra room to fit the USB plug and the servo wires through. In order to protect the wires from rubbing on the sharp edge of the hole, the first step was to sand down the hole to a nice smooth finish. In addition to that, we have placed an order for rubber grommets. The grommets are sized to fit into the 3/4 inch hole and leave us with enough room for the cables to fit and be protected from the wear and tear of vibration and rubbing against a metal edge. We have not yet received the grommets, so until they arrive, the wires will be placed through the hole as the image shows.

2.4 Positioning Servos & Testing the DC Motor

Another need that was addressed involved preparing the motors for correct operation when we are ready to test the complete robot. The two steps required involved proper positioning of the servo motors and mounting. In addition to that, testing and verification of the dc motor was completed.

To position the servo motors properly, we had to make some decisions about where the work envelope would be and what was most convenient for the PIC controller to operate everything successfully. The first and most important factor to consider is the work envelope for the servos. Each brand of servo has different specifications and as a result requires separate coding and conditions for achieving the same positions. Another important factor to consider is how positioning affects the control signal length and what is best for the bandwidth available to us to adhere to the necessary refresh rate of the servo motors as a group. With these constraints in mind, we set about positioning each of the four servo motors.

The first servo setup was for the upper grip. This is a Hi-Tec, HSR-5995TG model servo. We can achieve slightly more positional control than the 0 to 180 degrees that are normally provided with standard servos, however, for the upper grip, we require only 90 degrees of movement. The closed grip position will

Figure 4: USB Plug from Camera in Platform Hole

be aligned to 0 degrees. This decision was based on convenience as well as the fact that it uses the least amount of time for the PIC controller. The zero degree position corresponds to approximately 1100μS. The reason that time is a concern for the PIC controller is that if each servo is positioned close to its maximum (180 degrees) this will take close to 2 milliseconds for each. Adding up the time for four servos reaches 8 milliseconds and if any additional operations are occurring that the PIC needs to control then additional time is consumed and this makes it difficult to stabilize the servos since each of them require a refresh rate somewhere between 12-26 milliseconds (varies with each servo). Going outside this range has the adverse effect of either losing the position of where the servo should be (with potentially disastrous side-effects), or erratic movements and draining the batteries prematurely (also potentially disastrous side-effects). The ninety degree position of the Hi-Tec servo settled at about 1.5 milliseconds, which is helpful in reducing the amount of time required for the PIC to process all of its instructions. To position the servo properly, a simple PIC program was written to position the upper servo in the 0 degree position and then the grip was moved until it was aligned properly with the servo, effectively 'zero-ing' the upper grip into alignment.

Next, the lower grip servo was calibrated. The type of servo motor used was a Hobbico CS-35 which operates with slightly different parameters from the other servo motors. The work envelope of the lower grip in the closed position of the Hobbico servo was set to 90 degrees. This allowed us to lower the position to zero degrees to open the lower grip completely. In addition to this, the length of the control signal would be at its smallest using this area of the work envelope. The control signals for the 90 degree position coincides with a 1.51 milliseconds length and the 0 degree position takes 0.7 milliseconds. Aligning the position of the lower grip was done similarly to the upper grip. Now that the servos have been calibrated, the grip is ready to go.

The next servo to calibrate was the pan servo for the camera (Futaba S3003). The Futaba servo for the camera pan operation was only able to provide us with 0 to 164 degrees. This limitation is based upon

the amount of time (in milliseconds) required to achieve these positions. The zero degree position coincides with 0.34 milliseconds and the 164 degree position coincides with 2.06 milliseconds for control pulse length. Although this isn't the complete range that we were hoping for (180 degrees), it is sufficient to pan around and gather a reasonably wide range of available pictures. This can be altered at a later date as required through the addition of a different servo for the camera pan servo or possibly altering the maximum allowed time for a control signal (over 2mS). To align the camera pan servo, we placed the zero point facing left of the forward traveling direction under the platform. Using 164 degrees of rotation, the camera can almost turn exactly opposite the direction it starts from. This provides a reasonably wide range of view from the panning perspective of the camera. One problem that occurs with wired devices is that cords get tangled up when panning. This is the main reason why we decided not to do a hardware modification to the servo for continuous rotation. With the pan servo calibrated properly, we have the capability to swivel the camera to view Alice II's surroundings when moving in the direction that the camera is currently mounted.

Since the camera can only see in half of its possible views, the tilt servo was added to improve upon the angles of vision. The camera servos are mounted directly to the platform without any hardware (2-pound foam tape), this results in a limited range of tilt available before the platform contacts the camera. We decided to restrict the tilting movement to 90 degrees, similar to the line grip servos. In addition to that, the servo used was the Hobbico CS-35. As a result, the software calibration had already been completed, once the servo was aligned at the zero degree point (with the camera facing straight towards the ground), it provided a 90 degree rotation down to parallel with the platform. This gives nearly a complete quarter of a sphere for the view to gather images for processing.

The last item to test and verify was the dc motor. We needed to order a special mounting plate (discussed in the previous set of notes) to fit the dc motor into the same space as the original part used. Once the motor was mounted, as shown in figure 5, it was wired up to the h-bridge and we were able to drive the gear train via the PIC control board. The previous dc motor that was used was not a gear head motor unlike the current Sanyo motor does. The difference in torque available is substantial and from some preliminary tests (we added friction to the wheels), the new small motor performed very well. An additional image is included to provide a better perspective of the size and interface that the new motor has with the existing power train.

With each of the servos positioned correctly and the new dc motor tested and verified, the bulk of the physical construction has been completed. There are still a few remaining items to be done as discussed at the end of the report for what remains to be done.

2.5 TS-5500 Code

Chris has been working on providing us with an interface between the PIC control board and the TS-5500 as well as a software interface to the camera so that we can see still shots. Code was provided by Maciej for the camera which gave the basis for the simple interface that we wanted. The first stage of development was to put together a simple response system to deal with sensor inputs from the line grip and issue commands accordingly based on a simple preliminary behaviour. The next stage of development from the coding phase using the TS-5500 was to obtain images with the camera and be able to view them remotely verifying proper operation.

At present, we have a simple behaviour plan for what to do in the event that obstacles are encountered. Since we have contact switches as our main set of sensors at present, the robot is unable to take evasive action before it physically encounters the obstacles. Work is in progress to add extra sensing to allow evasive actions before contact occurs. However, for now, the chart below demonstrates the preliminary setup for the robot's behaviour. There are two parts to the flow diagrams covering the rudimentary behaviour that we have planned for the line crawler at this stage of development.

The preliminary behaviour is a slightly less complicated version of what will eventually become the behaviour of Alice II. Once we have had a chance to experiment with additional sensors and refine the

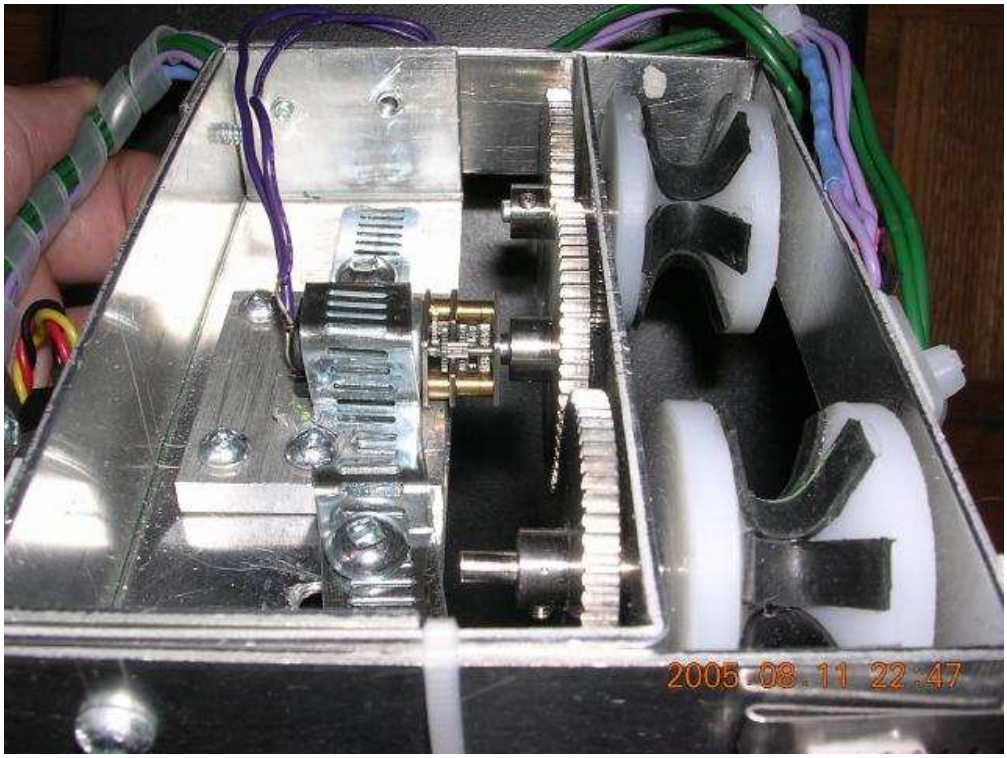

Figure 5: Top View of New DC Motor

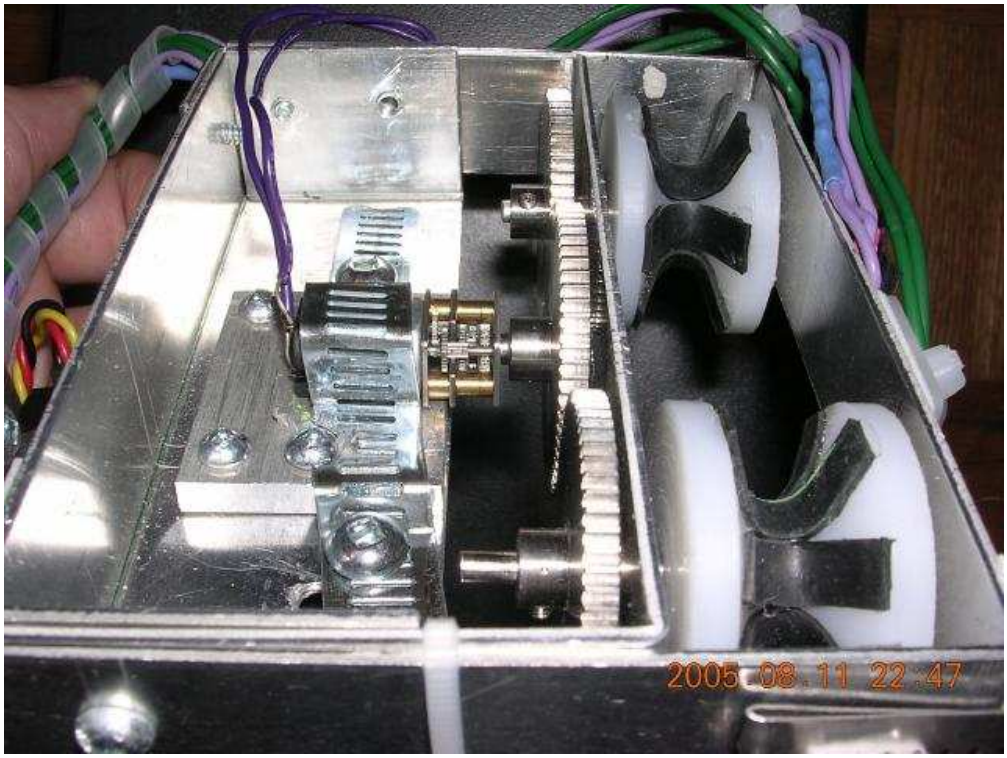

Figure 6: Side View of New DC Motor

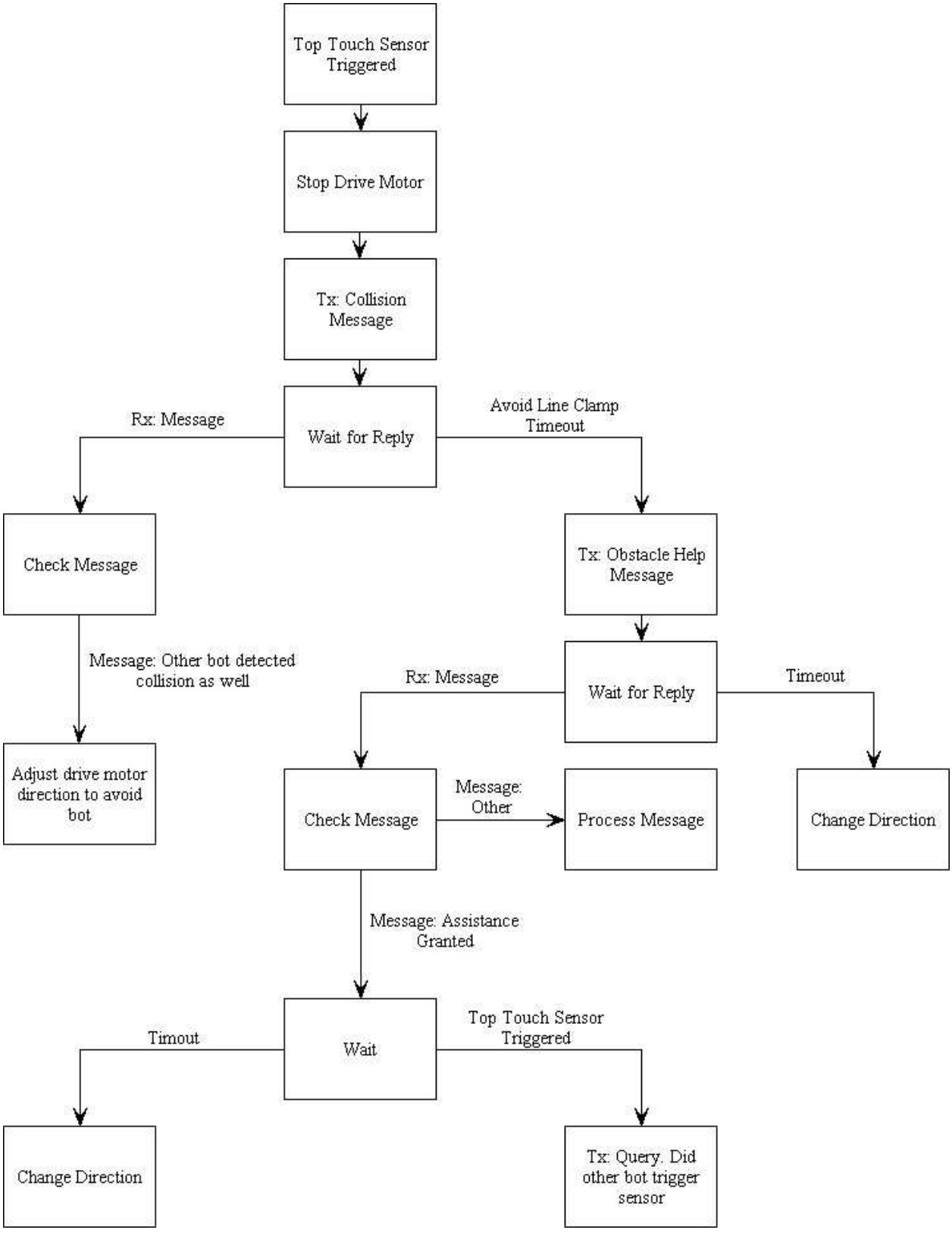

Figure 7: Main Behaviour Flow Diagram

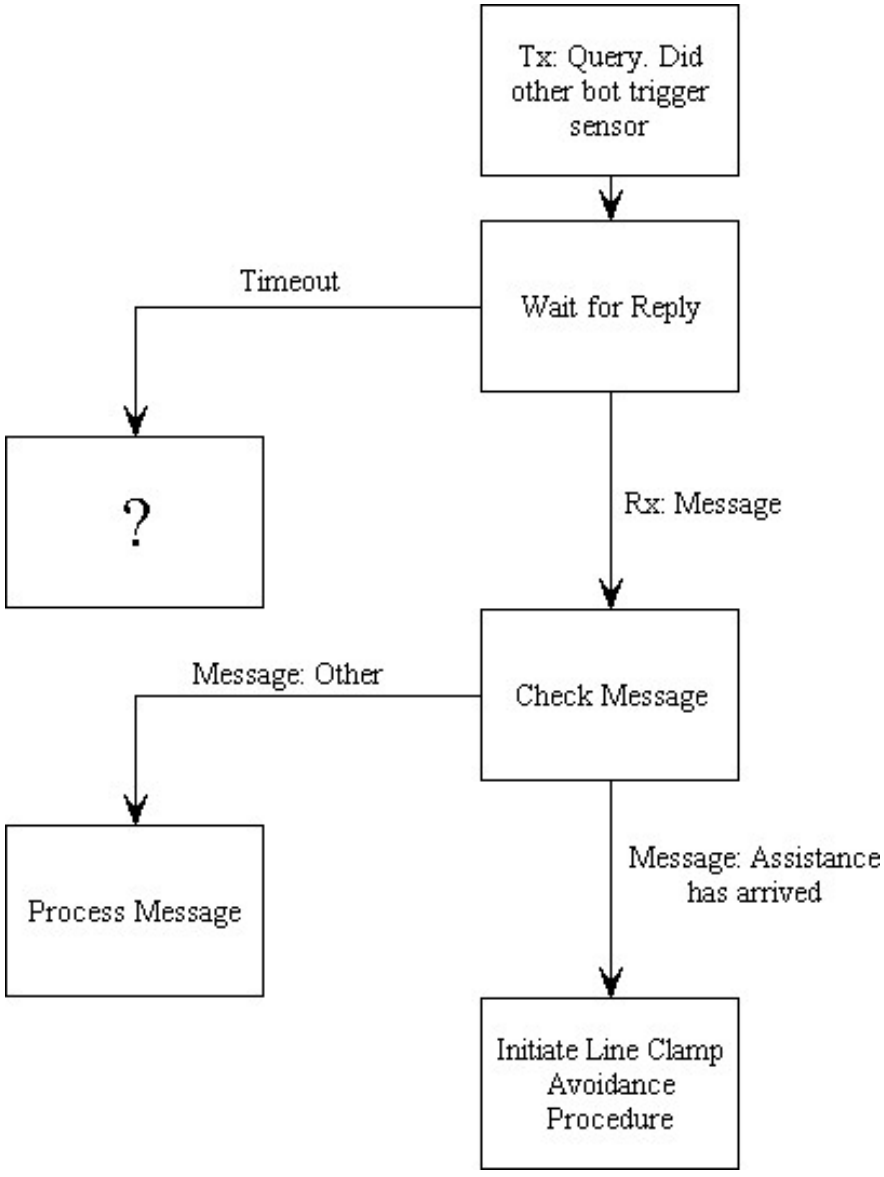

Figure 8: Additional Behaviour Flow Diagram

current behaviour, we plan to create a less reactive and more proactive system.

In addition to the rudimentary behaviour for interfacing the TS-5500 to the rest of the system, Chris has been working on getting the camera to take some still images and have them available to an external machine for viewing. Initially, the work involved was deciphering the existing code that Maciej provided for us and shape it more for running a demonstration of the line crawler's capabilities. Our preliminary plan is to roll along the sky wire until we reach an obstacle, back off slightly and look using the camera in the general direction of where the insulators would be located (adjust the camera position as required), take a picture or two and then relay them via wireless ftp to a remote machine for examination. The code is nearly complete, with some minor revisions and streamlining currently taking place (there was support for streaming video, which we aren't planning on using right away since it uses up a lot more battery power than just grabbing still images).

2.6 Construction Status

The current construction status for the two line crawler robots are reasonably close to completion. The first line crawler is completed with the exception of the servo extension wires (on order) and the addition of the grommet for the camera and its servo control wires. The second line crawler is nearly completely built, the sensors need to be added and wired up. Similar to the first bot, the servo extension wires and the grommet are on back order. Once the orders are received we should be all set to finalize the first bot and the second will follow shortly afterwards. This will provide us with two separate working line crawlers.

3 What's Next?

The next few steps in the line crawler project are working towards getting the experimental setup running and troubleshooting any potential pitfalls we may have. The immediate problem is to take care of final construction once the remaining materials arrive. Hopefully this will occur at some point early next week (the week beginning August 15). Once the construction of the first bot is complete, preliminary experimental trials will begin at my apartment. All of the towers and wire are here as well as the obstacles for the line. For experimental trials we will begin by doing simple locomotion tests to ensure that the design works how we wanted it to with all of the weight attached, moving it around to balance the line crawler properly. Once that has happened, then we will be doing some more interesting experiments to discover the performance of the camera system and how to coordinate all of the devices to acquire images of specific objects in the apartment. This will eventually lead to outdoor experiments where some of the elements come into play, especially wind and perhaps rain. Peter will be continuing with construction of the second bot since there is still more work to be done assembling sensors and wiring all of the devices. This should set us up nicely for even more advanced experiments including having the bots communicating with each other for problem resolution.

A Appendix A: CC5X Code

A.1 Control1_1.c

/*

This is a control interface to respond to the TS-5500 and drive all of the devices with the PIC. The preliminary version of this control software will manage two servo motors and the DC motor drive. This will allow us to move forward and backwards on the line as well as open and close the grip mechanism.

The protocol that we are using involves a single byte, sent serially at a rate of 9600bps to the serial input on the PIC. The PIC takes the byte, decodes it and controls the specified device in the manner provided. The first three bits dictate the device to be controlled and the last 5 bits specify the instruction accompanying the device control. To begin with, the devices managed will be as follows: 000 = DC Motor, 001 = Upper Grip Servo, 002 = Lower Grip Servo
The instructions corresponding to each motor is as follows:
DC Motor:
00000 = Turn motor clockwise
00001 = Turn motor counter clockwise
00010 = Apply brakes to motor
00011 = Put motor in quiescent state (off)

Upper Servo:
00000 = Open grip to 90 degree position (HiTec Servo)
00001 = Close grip to 0 degree position (HiTec Servo)

Lower Servo:
00000 = Open grip to -90 degree position (Hobbico Servo)
00001 = Close grip to 0 degree position (Hobbico Servo)

At present, this is all that will be included in the first control interface. The next step is to add extra possibilities for the servo motors. Also, we will be adding two extra servos for the camera control. These will have slightly different control schemes since we won't be as worried about motion control when moving the camera as we are with moving the entire grip. The difference will be in the step size that we move the servos.

Author: Dan Lockery
Date: July 10, 2005
Version: 1.0

Revision: 1.1
Date: July 26, 2005
Changes:
The changes that are being included in the first revisions are to add the control required for operating the camera servos. This will finalize the

positions of the servos (servo 1, 2, 3 and 4 will all be assigned individual
tasks of operating grip or camera joints). The servos will be as follows:
1). Upper grip servo
2). Lower grip servo
3). Pan servo for camera
4). Tilt servo for camera
Support will be provided for the camera movement in varying step sizes,
the protocol calls for +/- 16 steps using a signed 5-bit number. Knowing
where the servos are positioned is very important and support must be provided
for it.

*/

```
#include c:/Program Files/cc5x/16F871.h // This is the device header file
#include c:/Program Files/cc5x/minid.c // This is the delay file
#include c:/Program Files/cc5x/Delay.c // contains the millisecond delay

void main()


// First, set up the serial communication link.
RCSTA = 144; // assign RCSTA 90h
TXSTA = 36; // enables 8-bit asynch communication with high speed BRG
SPBRG = 25; // this sets the baud rate to 9600bps since BRG = 1
TXIE = 0; // this keeps the interrupts disabled
PORTC = 0b.0000.0000;
TRISC = 0b.1000.0000; //0 = Output, 1 = Input

// Next, set up PortB as outputs since they will run the motors!
PORTB = 0b.0000.0000;
TRISB = 0b.0000.0000; // This configures port B as all outputs

uns8 pos1; // position of the upper servo motor
uns8 pos2; // position of the lower servo motor
uns8 pos3; // position of the pan camera servo motor
uns8 pos4; // position of the tilt camera servo motor
uns8 vpos1; // desired position of the upper servo
uns8 vpos2; // desired position of the lower servo
uns8 vpos3; // desired position of the pan camera servo
uns8 vpos4; // desired position of the tilt camera servo
uns8 spos1; // starting position of the upper servo
uns8 spos2; // starting position of the lower servo
uns8 mov3; // movement desired for the camera pan servo
uns8 mov4; // movement desired for the camera tilt servo
uns8 dir3; // direction of movement for the camera pan servo
uns8 dir4; // direction of movement for the camera tilt servo
uns16 del; // delay for repeating ctrl signals.
uns8 device; // device that we are going to be altering
```

```
uns8 pos; // position of the servo that we want to change
uns8 instr; // instruction for what to do based on the device!

PORTB.2 = 0; // initialize the DC motor in quiescent mode
PORTB.3 = 0; // quiescent mode now achieved.
PORTB.4 = 0; // start servo motor ctrl signal in low position
PORTB.5 = 0; // start servo 2 in low position
PORTB.6 = 0; // start servo 3 in low position
PORTB.7 = 0; // start servo 4 in low position

del = 16; // loop delay for this many milliseconds
dir3 = 0; // arbitrarily set to negative direction
dir4 = 0; // arbitrarily set to negative direction
mov3 = 0; // set to zero to begin with
mov4 = 0; // set to zero to begin with
pos1 = 136; // position the upper grip servo at 0 degrees
pos2 = 187; // position the lower grip servo at 90 degrees
pos3 = 45; // start the camera pan servo at 0 degrees
pos4 = 87; // this starts the camera tilt servo at 0 degrees
vpos1 = 136; // to begin with these will be matched
vpos2 = 187; // with their pos counterparts
vpos3 = 45; // match position with these also
vpos4 = 87; // match position for the tilt as well
spos1 = 0; // indicates that this is closed to begin with
spos2 = 0; // indicates that this is closed to begin with

// Note that the positions are all in the first half
// this minimizes the amount of time that the PIC is busy.

while(1)  // infinite loop for what to do

// first, send out starting signals to the servo motors
pos = pos1; // load value to set for servo 1
PORTB.4 = 1; // set servo 1 output to high
delayVar(pos); // wait for a set amount of time
PORTB.4 = 0; // set control pin low
// Next, set up the second servo motor (lower grip)
pos = pos2; // load up value to set for servo 2
PORTB.5 = 1; // set servo 2 output high
delayVar(pos); // wait for a set amount of time
PORTB.5 = 0; // set control pin low
// Next, set up the camera pan servo motor
pos = pos3; // load up value to set for servo 3
PORTB.6 = 1; // set servo 3 output high
delayVar(pos); // wait for a set amount of time
PORTB.6 = 0; // set control pin low to finish the pulse!
// Next, set up the camera tilt servo motor
pos = pos4; // load up value to set for servo 4
```

```
PORTB.7 = 1; // set servo 4 output high
delayVar(pos); // wait for a set amount of time
PORTB.7 = 0; // set control pin low to finish the pulse!

// Once the servos are set, decode incoming control byte

if(RCIF)  // when we get a byte serially
nop(); // settling time
instr = RCREG;
if(instr.0 == 0 && instr.1 == 0 && instr.2 == 0)  //DC motor
if(instr.3 == 0 && instr.4 == 0)  // turn motor off
PORTB.2 = 0; // set 'In1' to 0 for DC motor off
PORTB.3 = 0; // set 'In2' to 0 for DC motor off
// end of inner if for clockwise motor turning
else if(instr.3 == 1 && instr.4 == 0)  // hit the brakes
PORTB.2 = 1; // set 'In1' to 1 for braking
PORTB.3 = 1; // set 'In2' to 1 for braking
// end of inner if for CCW motion
else if(instr.3 == 0 && instr.4 == 1)  // turn motor on CW
PORTB.2 = 0; // set 'In1' to 0 for CW motion
PORTB.3 = 1; // set 'In2' to 1 for CW motion
// end of motor on CW
else if(instr.3 == 1 && instr.4 == 1)  // turn motor on CCW
PORTB.2 = 1; // set 'In1' to 1 for CCW motion
PORTB.3 = 0; // set 'In2' to 0 for CCW motion

else;
// faulty command so don't do anything
// end of what to do when selecting DC motor
else if(instr.0 == 1 && instr.1 == 0 && instr.2 == 0)
if(instr.3 == 0 && instr.4 == 0 && instr.5 == 0 && instr.6 == 0 && instr.7 == 0)

// open the upper grip
vpos1 = 190; // this is equal to the open position
spos1 = 0; // starting position is meant to be from the closed position
// end of inner IF for the open upper grip case
else if(instr.3 == 1 && instr.4 == 0 && instr.5 == 0 && instr.6 == 0 && instr.7 == 0)  // close upper grip
vpos1 = 136; // this is equal to the closed position
spos1 = 1; // starting position is meant to be from the open position

else ; // else, an incorrect command was issued
// end of else if for controlling RC servo motor #1
else if(instr.0 == 0 && instr.1 == 1 && instr.2 == 0)  // the 2nd servo
if(instr.3 == 0 && instr.4 == 0 && instr.5 == 0 && instr.6 == 0 && instr.7 == 0)  // open lower grip
vpos2 = 87; // this coincides with a 0 degree open position for servo2
spos2 = 0; // this means that the starting position was closed
// end of inner if for
else if(instr.3 == 1 && instr.4 == 0 && instr.5 == 0 && instr.6 == 0 && instr.7 == 0)  // close lower grip
```

```
vpos2 = 187; // this is equal to the closed position
spos2 = 1; // indicates started from the open position

else; // otherwise, there was an error in the command


else if(instr.0 == 1 && instr.1 == 1 && instr.2 == 0)  // pan camera servo
//the next part of the code deciphers how much to move the servos
if(instr.7 == 1)  // here is what to do for negative movement
dir3 = 0; // signifies the direction is negative
mov3 = 0; // clear the movement counter before generating a new count
if(instr.6 == 0)
mov3 = mov3 + 8; // add 8 steps of movement for a bit in the fourth position
if(instr.5 == 0)
mov3 = mov3 + 4; // add 4 steps of movement for a bit in the third position
if(instr.4 == 0)
mov3 = mov3 + 2; // add 2 steps of movement for a bit in the second position
if(instr.3 == 0)
mov3 = mov3 + 1; // add 1 step of movement for a bit in the first position
mov3 = mov3 + 1; // this is added since the binary range runs from 0 to 15

// Next, we need to check and see if we can move negatively
if(pos3 - mov3 ≤ 45) // this implies we are in trouble
mov3 = pos3 - 45; // this sets up mov3 to turn us to 0 degrees
// Finally, assign the virtual position the location that we want to move to.
vpos3 = pos3 - mov3; // vpos now assigned for pan camera servo

// Now we are setup to move in the negative direction

// end if for negative movement of pan camera servo
else  // here is what to do for positive movement
dir3 = 1; // signifies the direction is positive
mov3 = 0; // clear the movement counter before generating a new count
if(instr.6 == 1)
mov3 = mov3 + 8; // add 8 steps of movement for a bit in the fourth position
if(instr.5 == 1)
mov3 = mov3 + 4; // add 4 steps of movement for a bit in the third position
if(instr.4 == 1)
mov3 = mov3 + 2; // add 2 steps of movement for a bit in the second position
if(instr.3 == 1)
mov3 = mov3 + 1; // add 1 step of movement for a bit in the first position
mov3 = mov3 + 1; // this is added since the binary range runs from 0 to 15

// Next, we need to check and see if we can move positively
if(pos3 + mov3 ≥ 255) // this implies we are in trouble, adjust the movement
mov3 = 255 - pos3; // this sets up mov3 to turn us to 164 degrees
// Finally, assign the virtual position the location that we want to move to.
vpos3 = pos3 + mov3; // vpos now assigned for pan camera servo
```

```
// Now we are setup to move in the positive direction

// end else for positive movement of pan camera servo

// end of controlling the pan camera servo motor

// Next up is the tilt camera servo motor control routine

else if(instr.0 == 0 && instr.1 == 0 && instr.2 == 1)  // tilt camera servo
//the next part of the code deciphers how much to move the servo
if(instr.7 == 1)  // here is what to do for negative movement
dir4 = 0; // signifies the direction is negative
mov4 = 0; // clear the movement counter before generating a new count
if(instr.6 == 0)
mov4 = mov4 + 8; // add 8 steps of movement for a bit in the fourth position
if(instr.5 == 0)
mov4 = mov4 + 4; // add 4 steps of movement for a bit in the third position
if(instr.4 == 0)
mov4 = mov4 + 2; // add 2 steps of movement for a bit in the second position
if(instr.3 == 0)
mov4 = mov4 + 1; // add 1 step of movement for a bit in the first position
mov4 = mov4 + 1; // this is added since the binary range runs from 0 to 15

// Next, we need to check and see if we can move negatively
if(pos4 - mov4 ≤ 87) // this implies we are in trouble, adjust the movement
mov4 = pos4 - 87; // this sets up mov4 to turn us to 0 degrees

// Finally, assign the virtual position the location that we want to move to
vpos4 = pos4 - mov4; // vpos now assigned for tilt camera

// Now we are setup to move in the negative direction

// end if for negative movement of tilt camera servo
else  // here is what to do for positive movement
dir4 = 1; // signifies the direction is positive
mov4 = 0; // clear the movement counter before generating a new count
if(instr.6 == 1)
mov4 = mov4 + 8; // add 8 steps of movement for a bit in the fourth position
if(instr.5 == 1)
mov4 = mov4 + 4; // add 4 steps of movement for a bit in the third position
if(instr.4 == 1)
mov4 = mov4 + 2; // add 2 steps of movement for a bit in the second position
if(instr.3 == 1)
mov4 = mov4 + 1; // add 1 step of movement for a bit in the first position
mov4 = mov4 + 1; // this is added since the binary range runs from 0 to 15

// Next, we need to check and see if we can move positively
```

```
if(pos4 + mov4 ≥ 187) // this implies we are in trouble, adjust the movement
mov4 = 187 - pos4; // this sets up mov4 to turn us to 90 degrees
// Finally, assign the virtual position the location that we want to move to
vpos4 = pos4 + mov4; // vpos now assigned for tilt camera

// Now we are setup to move in the positive direction

// end else for positive movement of tilt camera servo

// end of controlling the tilt camera servo motor
else;
// else, at the moment, do nothing

// end of what happens when a byte is received serially

delayMs(del); // pause for 16 milliseconds

// before loop back and position servos, test if position changed
// first check servo 1, or the upper grip servo position.
if(pos1 != vpos1 && spos1 == 0)  // this implies we need to open the upper grip
pos1++; // increment the position by one towards the desired direction!
// end of if for moving towards an open grip command!
else if(pos1 != vpos1 && spos1 == 1)  // implies we need to close upper grip
pos1–; // decrement the position by one towards the desired direction!
// end of if for moving towards a closed grip command for the upper or first servo
else; // do nothing with servo1

// next, check servo 2, or the lower grip position
if(pos2 != vpos2 && spos2 == 0)  // this implies open the lower grip
pos2–; // decrement the position by one to move towards the desired direction
// end of if for moving towards an open grip position for servo 2
else if(pos2 != vpos2 && spos2 == 1)  // close the lower grip
pos2++; // increment position by one step to move towards the desired direction
// end of if for moving towards a closed grip position for servo 2

// Next, include processing for pan and tilt servos
if(pos3 != vpos3 && dir3 == 0)  // pan in a negative direction
pos3–; // decrement position by one
// end if
else if(pos3 != vpos3 && dir3 == 1)  // pan in a positive direction
pos3++; // increment position by one
// end of else if

// Finally, include support for the tilt camera servo
if(pos4 != vpos4 && dir4 == 0)  // tilt in a negative direction
pos4–; // decrement position by one
// end of if
else if (pos4 != vpos4 && dir4 == 1)  // tilt in a positive direction
```

```
pos4++; // increment position by one
// end of else if

else; // do nothing

// end of while loop, go back and loop until done!
```