VISUAL TARGET TRACKING SYSTEM

By

Timothy Bagnull, Undergraduate ECSE ‘03
James Deloge, Graduate EE ‘03
Chad Helm, Graduate EE ‘04
Matthew Sked, Graduate EE ‘04

ECSE-4962 Senior Design Project
Project Advisor: Professor John T. Wen
Lab TA: Ben Potsaid

April 30, 2003
Rensselaer Polytechnic Institute
Abstract

This report documents the development, design and testing of a vision tracking pan-tilt control system. Modeling, simulation, and control of a pan-tilt system will be discussed. Vision tracking based upon a simple edge detector will be used as feedback to the controller. Beginning with a basic pan-tilt system controls engineering analysis is applied to design a controller capable of meeting initial system specification. To locate the moving target computer vision is used implementing a Robert’s Edge Detector and Incremental Step Function for the purpose of initially locating the target and then tracking it. While the design was successful in tracking a moving target using computer vision, the final system fell short of the projects initial specifications. Future considerations for improvement of the system are discussed.
# TABLE OF CONTENTS

1. Introduction ..................................................................................................................... 5  
   1.1 Assumptions .............................................................................................................. 6  
   1.2 Initial System Specification ...................................................................................... 7  
      1.2.1 Control ............................................................................................................... 7  
      1.2.2 Computer Vision ................................................................................................ 7  
2. Design Methodology ....................................................................................................... 8  
   2.1 Mechanical System ................................................................................................... 8  
      2.1.1 Approach ............................................................................................................ 8  
   2.2 Computer Vision System ........................................................................................ 13  
      2.2.1 Approach .......................................................................................................... 13  
3. Design Details ............................................................................................................... 17  
   3.1 Control .................................................................................................................... 17  
      3.1.1 Linear Modeling ............................................................................................... 17  
      3.1.2 Friction Identification ...................................................................................... 21  
      3.1.3 Simulations with Friction ................................................................................. 22  
      3.1.4 Real-Time Controller Design ........................................................................... 24  
      3.1.5 Controller Performance .................................................................................... 26  
      3.1.6 PID Controller .................................................................................................. 28  
   3.2 Vision ...................................................................................................................... 29  
      3.2.1 Roberts Edge Detector ..................................................................................... 30  
      3.2.2 Edge Enhancement ........................................................................................... 30  
      3.2.3 Edge Localization ............................................................................................ 32  
      3.2.4 Incremental Step Function ............................................................................... 32  
      3.2.5 Vision System Flowchart ................................................................................. 34  
   3.3 Camera Bracket ....................................................................................................... 35  
4. Design Verification ....................................................................................................... 38  
   4.1 Experimental Set-up .............................................................................................. 38  
   4.2 Verification ............................................................................................................. 39  
   4.3 System Performance ............................................................................................... 40  
5. Problems Encountered .................................................................................................. 41  
6. Cost and Schedule ......................................................................................................... 42  
   6.1 Cost Analysis .......................................................................................................... 42  
      6.1.1 Pan-Tilt Mechanism ......................................................................................... 42  
      6.1.2 Computer Vision .............................................................................................. 42  
      6.1.3 Labor Cost ........................................................................................................ 43  
   6.2 Schedule .................................................................................................................. 43  
7. Professional and Societal Considerations ..................................................................... 44  
8. Future Modifications ..................................................................................................... 45  
   8.1 Kalman Filter ........................................................................................................... 45  
   8.2 Pattern Recognition ................................................................................................. 47  
   8.3 General Improvements ............................................................................................ 47  
9. Conclusions ................................................................................................................... 48  
   9.1 Results ..................................................................................................................... 48  
   9.2 Multidisciplinary Aspect of the Project .................................................................. 48
# 10. Individual Team Member Contributions

- **10.1 Tim Bagnull**
- **10.2 James Deloge**
- **10.3 Chad Helm**
- **10.4 Matthew Sked**

# 11. Final Report Authorship

- **11.1 Tim Bagnull**
- **11.2 James Deloge**
- **11.3 Chad Helm**
- **11.4 Matthew Sked**

# 12. References

- Appendix 1: Pan System Geometry
- Appendix 2: Tilt System Geometry
- Appendix 4: Group Member Resumes
- Appendix 5: testpath.cpp
- Appendix 6: Kalman Filter Algorithm in Visual Basic
- Appendix 7: grabfiltermotion.cpp
- Appendix 8: pantiltinitdesign.m
- Appendix 9: bodyawithmotor.m
- Appendix 10: bodybwithcamera.m
- Appendix 11: compositebodies.m
- Appendix 12: gravitylin.m
- Appendix 13: masscenter.m
- Appendix 14: massmatrix.m
- Appendix 14: pantiltlin.m
- Appendix 15: parallelaxis.m
- Appendix 16: sysmodcompare.m
- Appendix 17: tstats.m
- Appendix 18: SolidWorks Drawings
- Appendix 19: CAD Drawings
# TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>System Flow Diagram Model</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Geometry Used to Determine Angular Speed Goal</td>
<td>6</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Root Locus of the Pan Axis</td>
<td>18</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Position and Torque Step Response for Pan Axis</td>
<td>19</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Root Locus of the Tilt Axis</td>
<td>20</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Position and Torque Step Response for Pan Axis</td>
<td>21</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Non-Linear Friction Simulations of Pan</td>
<td>23</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Non-Linear Friction Simulations of Tilt</td>
<td>24</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Pan Axis ARCS model with controller and friction compensation</td>
<td>25</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Tilt axis ARCS model with controller and friction compensation</td>
<td>26</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Pan Simulation and System Comparison</td>
<td>27</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Tilt Simulation and System Comparison</td>
<td>28</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Non-Linear Simulation and PID controller</td>
<td>29</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Edge Enhancement</td>
<td>31</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Edge Enhancement – Poor Lighting</td>
<td>31</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Edge Enhancement – Focus Effects</td>
<td>32</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Partition of image into four quadrants</td>
<td>33</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Vision System Flowchart</td>
<td>34</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Original Web Cam Mount</td>
<td>35</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Camera Mount</td>
<td>36</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Camera Mount with Spacer</td>
<td>37</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Gant Chart</td>
<td>43</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Pan System Geometry</td>
<td>52</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Tilt System Geometry</td>
<td>53</td>
</tr>
<tr>
<td>Figure 25</td>
<td>SolidWorks Drawing - Camera Mount Spacer</td>
<td>92</td>
</tr>
<tr>
<td>Figure 26</td>
<td>SolidWorks Drawing - Camera Mounting Bracket</td>
<td>93</td>
</tr>
<tr>
<td>Figure 27</td>
<td>SolidWorks Drawing - Camera</td>
<td>94</td>
</tr>
<tr>
<td>Figure 28</td>
<td>SolidWorks Drawing - Camera with Mount</td>
<td>95</td>
</tr>
<tr>
<td>Figure 29</td>
<td>SolidWorks Drawing - Vision Tracking Pan/Tilt System</td>
<td>96</td>
</tr>
<tr>
<td>Figure 30</td>
<td>CAD Drawing – Camera Mount Buffer</td>
<td>97</td>
</tr>
<tr>
<td>Figure 31</td>
<td>CAD Drawing – Camera Mounting Bracket</td>
<td>98</td>
</tr>
</tbody>
</table>
1. Introduction

The inspiration for this project came from the group’s desire to create an automated system that implements concepts that are active areas of research. Two of the group members had previously taken a course in Computer Vision that both found fascinating and stimulating. They felt there could be a natural incorporation between Computer Vision and Control Systems and were prepared to demonstrate it.

The goal of this project will be to integrate a camera with the skeletal pan-tilt stages using feasible motors and drive-train to track the movement of a point. Upon initialization with the user interface the visual target tracking system will find the shape via an edge detection method.

A block diagram in Figure 1 shows the interaction of the different physical components (camera, frame grabber, CPU, ARCS card, and pan-tilt stages) with different software or physical connections (drivers, C++, ISA bus, SIMULINK, and physical motion). The diagram shows that the system is closed loop and will be able to operate solely on its own. As shown below, C++ will be used to interface between the vision drivers with the CPU. From the CPU we can send data via the ISA bus to the ARCS card. The software on the ARCS card in turn creates motion on the pan/tilt.

![System Flow Diagram Model](image)

**Figure 1: System Flow Diagram Model**

The target will be simulated through a MICROSOFT POWERPOINT presentation. As the point moves across the screen, the visual target tracking system will follow the target and attempt to place the point in the center of the field of view of the camera. The system will be able to track a shape moving up to 300 mm/s (approx. 1 ft/s) at a distance of 3 m (approx. 10 ft) from the screen.
1.1 Assumptions

To find these initial specifications we need to make an assumption about the optimal speed the point will move so that our system can track it. We decided the point will move about 300 mm/s (about 1 ft/s). This assumption was an educated guess based on what the optimal speed at which the CPU can process the data, the optimal speed we can communicate with the ARCS card and our assumption that our system will be 10 ft from the point we are tracking. This can be translated to a 6 deg/s angular velocity of our pan-tilt for both axes by using the geometry below:

![Figure 2: Geometry Used to Determine Angular Speed Goal](image)

Further hardware specifications must be made about the work envelope for our pan-tilt system. Again this can be calculated from the assumption that we will be about 10 ft from the screen. Also, we take into consideration the size of the screen which will be the limited area our target will reside in. Specifically the size of the screen is 1.52m x 2.03m. By solving a simple geometry problem we can solve that the system can move 18 degrees off center for the pan and 14 degrees off center for the tilt. In Appendix 1 are two drawings that show the geometric configuration of the system and the screen.
1.2 Initial System Specification

1.2.1 Control

The motion control specifications are as follows:
- Top angular motion of 6 degrees/s. for both axis.
- For acceptable reliability, the system must be no more than about 20 degrees off center of the screen.
- Desired settling time of 0.1 seconds
- Desired overshoot of 1% for tilt, 2% for pan.

A 1% and a 2% overshoot correspond to linear displacement on the screen of 6.1 mm and a 12.2 mm, respectively.

1.2.2 Computer Vision

The user interface will be designed in Visual Basic. The interface will include button commands for system initialization and edge detection. There will also be a display window depicting the image obtained by the camera. As the camera tracks the shape the display window will show the Kalman filter algorithm tracking the position. The interface will also contain a button for system shutdown, and other various debugging tools if needed.

The software also has certain specifications that must be made for the system to operate correctly. First the camera must not be stationed more than 20 degrees off from the center of the screen. This may cause occlusion of the point from the view of the camera or the point may become skewed which may make it difficult for the program to continue to track it. Next before we even begin the tracking algorithm we have an initialization procedure where we use edge detection to find our point that we need to track. At most we would like this procedure to take no longer than 5 seconds just so it is practical for use. Finally to increase our processing time, we would like to handle 12 frames/sec. This means we will use about every third sampled frame from our frame grabber. Discarding this many frames should not be a problem as long as the point does not move too fast so that the distance between the point in one frame and the next frame analyzed are not too far apart so that the Kalman Filter does not lose the point.
2. Design Methodology

2.1 Mechanical System

2.1.1 Approach

To design the physical mechanical system and tune a controller for the motion control of the system our team used a systematic and incremental approach. Before any control theory could be applied to the system the group first had to establish what the physical system was and then from this physical model create a mathematical representation with which control theory could be applied. Of course, there was a great deal of re-design in this project and because of this multiple trials had to be completed before the final project was finished. Included below are the specific milestones that were completed in the design process and what important lessons were taken away.

Physical Representation of the Pan-Tilt System I

What was involved?

This process involved selecting motors and pulleys with which to learn about the steps involved to first mathematically define the mechanical system and then begin creating a controller for that system. In addition, to verify that the parts selected would be able to meet the design specifications.

What was completed?

To quickly become familiar with the process involved our team decided it would be better to not waste time scrutinizing parts, as we would later learn how to better select these anyhow. For that matter, we based out initial design off the Course Teaching Assistant, Ben Potsaid’s example design. The only addition that was added to the system consisted of the camera mount that was need to how our sensor. Manually taking measurements of the system a mathematical representation was created. This mathematical representation represented our systems plant to which a controller was added and controls theory applied to create a controller. The controller created did meet our initial design specifications (without friction). The motors used for this first experiment were found to be feasible.

What was learned?

Firstly, our group learned the process of taking a physical mechanical system and converting it into a mathematical model. Secondly, the team became knowledgeable about the mechanics involved in a motor; how to tread a specifications sheet and how to
determine if a motor will be able to control a system. From the motor feasibility analysis we found that we could accomplish the same system with a lower torque tilt motor.

**Physical Representation of the Pan-Tilt System II**

*What was involved?*

This process involved selecting motors and pulleys with which to learn about the steps involved to first mathematically define the mechanical system and then begin creating a controller for that system. In addition, to verify that the parts selected would be able to meet the design specifications.

*What was completed?*

A new motor was selected for the tilt component of the system. Specifically, the team decided to use a Pittman PG6614 motor, which was a lower peak torque smaller profile motor compared to the Pittman GM8724S010 that was originally used. The only disadvantage of this motor was its cost which was found to be $154.16 compared to $89.80 that was the cost of the 8000 series motor. The system was re-defined and a new equation of motion was computed. From this new plant model a controller was created to meet the specifications. Both settled in .1 second and only the pan axis contained an overshoot of 2%, which was equal to the specification given.

*What was learned?*

The team was able to select an alternative motor and adapt the system to handle the new modification. Unfortunately, this specific motor was not able to be shipped in time and as a result the first motor configuration used was selected. On the other hand this problem was advantageous because of the time and money saved for the design process.

**Controller Tuning of the Pan-Tilt System Linear**

*What was involved?*

Linearalizing the system about a point of operation then putting the system into a linear model for controller tuning using MATLAB.

*What was completed?*

With the equations of motion defined and initial controller present in our system additional controller tuning was completed. The system was linearized using our maximum speed of 0.1 radians per second of target movement. This new system was then put into a linear model and using MATLAB’s *rltools* further controller tuning was completed.
What was learned?

The team learned how to linearize a system and how to create a linear model in SIMULINK. It is important to stress that friction was not considered for this phase of design.

**Controller Tuning of the Pan-Tilt System Non-Linear**

*What was involved?*

Implementing the system in a non-linear SIMULINK model to check the accuracy of the tuned linear model.

*What was completed?*

The linear model previously created was put into a non-linear model (that was provided by the class instructor Professor John Wen). The output of the system was plotted against the output of the linear model to analyze what further tuning had to be completed.

What was learned?

It was found that the linear model and non-linear model were alike but there were slight variations between the two.

**System Friction Identification**

*What was involved?*

Identifying friction by plotting the position of our system axis independently versus applied torque for the non-linear region of operation. Using these plots both coulomb and viscous friction variables can be identified.

*What was completed?*

The course Teaching Assistant, Ben Potsaid had previously created a MATLAB routine whose main function was to begin at a user defined torque and increment this torque while recording the position of the axis being analyzed until torque saturation was reached (which was also defined by the user). A second script was used (created by the course TA Ben Potsaid), which computed best fit trend lines on the plot to numerically compute coulomb and viscous friction values. To determine the start torque, a torque was applied to axis being analyzed, which the lowest value was found to result in noticeable motion of the axis. To determine the saturation torque the torque was increased until velocity remained constant with increasing torque. This procedure was completed for both the pan and the tilt axis and from this friction values were identified.
What was learned?

Fiction cannot be ignored. It has a significant effect on the performance of a system and needs to be taken into account when modeling any mechanical system. In addition, it is important to make sure the system has been “run-in” before reliable friction variable can be found.

**Controller Tuning of the Pan-Tilt System Linear and Non-Linear with Friction I**

*What was involved?*

This process involved recalculating the equation of motion with the experimentally determined friction variables then using this new system to design a controller.

*What was completed?*

The coulomb and viscous friction were added to the equation of motion. This new equation of motion resulted in the creation of a new system plant which linearized and put into the linear model. The controller was tuned using MATLAB’s rtool to meet the original specifications.

*What was learned?*

Friction will affect the dynamics of the system as well as the controller. Physically, this effect is very noticeable when implementing into the real-time system.

**Implementation of the Real-Time ARCS System**

*What was involved?*

Discretizing the system for a common sampling time and implementing this system into an ACRS SIMULINK model complete with friction compensation. Downloading the model onto the ARCS card and obtaining real-time data of the system.

*What was completed?*

With the final controller and system computer the C2D (continuous to discrete) command of MATLAB was used to put our model into a discrete time system format. A friction compensator was then created and the values that had previously been computed entered. With the final SIMULINK model defined MATLAB was used to create a compiled file that was able to be downloaded onto the ACRS board. While in ARCS real-time data was acquired to verify both the linear the non-linear simulations completed.

*What was learned?*
A step of 0.1 was commanded while in real-time ACRS control mode. From this commanded position it was found that the system took too long to settle to the desired position. This real-time data did not agree with our non-linear and linear simulations. The team decided that further investigation was needed to understand why these discrepancies existed in the system.

**Controller Tuning of the Pan-Tilt System Linear and Non-Linear with Friction II**

*What was involved?*

This process involved retuning the controller and checking the physical results using real-time data from the ARCS board.

*What was completed?*

The controller was retuned until the correct waveform of the physical system was obtained.

*What was learned?*

It was found that not enough initial torque was being given to the motors to result in a fast enough response. From further investigation it was found that the system performance was very sensitive to friction values. From the results that were given from the real-time data, it was decided by the group that either our friction variables were wrong or our system model was incorrect or a combination of both was resulting in the differences between simulated and physical systems. Time was fleeting at this point in design and the team decided to implement the design using what we currently had as more effort needed to be placed up on the vision section of our design.

**Controller Tuning of the Pan-Tilt System Linear and Non-Linear with Friction III**

*What was involved?*

In this section of the design process a PID (Proportional, Integral and Derivative) Controller was added to our system to see if increased system performance could be obtained.

*What was completed?*

A PID Controller was inserted into our SIMULINK model and a systematic approach to tuning the gains was completed. The team was able to successfully tune our gains to meet our initial specifications. The performance was verified using the real-time data acquired by the ACRS card.
What was learned?

It was found that our system with the friction estimates that we had defined could be tuned to meet initial specifications.

2.2 Computer Vision System

The mechanical pan-tilt system would not be useful without a sensor to acquire position data with. It was initially decided, by the team, that this information would be found using computer vision. To design the computer vision system, various tasks had to be completed. Firstly, a camera had to be found with a vision card so that image information could be acquired and processed. Secondly, the computer interfaces between the vision card as well as the ARCS motion card had to be researched and tested. Third the computer vision algorithms had to be implemented in computer code and combined with the vision and control interfaces. Included below are the specific milestones that were completed in the design process and what important lessons were taken away.

2.2.1 Approach

Initial Vision Design

What was involved?

Initial vision design involved deciding upon an algorithm to implement the vision data acquisition in, to decide what language to implement the code in and to choose a vision card to work with.

What was completed?

Initially, it was decided by the group members with experience in Computer Vision that the most efficient approach to track would be to find a target (initialize the system) using an edge detector algorithm and then track that target (target motion tracking) using a Kalman Filter. The mathematical algorithms were researched. The team also decided that it would be better to implement the vision code in Visual Basic (VB) because of the ease of making user interfaces in VB.

What was learned?

The mathematics behind the Robert’s Edge Detector and Kalman Filter.

Vision Card Interface

What was involved?
The team had to become acquainted with the vision card given to them by course instructor John Wen and learn how to grab and display images with the camera using VB.

What was completed?

The documentation for the vision card was reviewed and sample VB code (that was provided by the manufacturer) was run to learn about the coding instructions used for acquiring image data off the camera and analyzing the image data.

What was learned?

Interface to the MATROX Vision Card. Visual Basic commands for using the vision card.

Edge Detector Design in VB

What was involved?

Implementing the mathematical expression for the Robert’s Edge Detector into VB and processing verification of the algorithm using image data acquired from the camera.

What was completed?

The Edge Detector was coded in VB and an interface to the vision card was added. This resulted in a system that could successfully track a moving target based on edge detection alone. The only part that this system lacked was an interface between the data and the ACRS card.

What was learned?

VB programming for MATROX Card interface and Edge Detector. Processing acquired information to obtain XY coordinates needed for the motion control.

Interface to ARCS in VB

What was involved?

Reviewing the ARCS interface documentation and creating code capable of sending data to the ARCS card.

What was completed?

The ARCS card documentation specifies C++ as the programming language to use in interfacing to the card. The team believed that this could be done in VB as well thereby fitting together with the MATROX Card interface and vision algorithms. Adapting the interface code to VB was unsuccessful.
What was learned?

The interface to the ARCS card cannot be completed in VB and making an interface in C++ just for this section, would have resulted in difficulties merging the other code. The ACRS interface was a top priority over the vision data. If the vision data was unable to be finished the team could still test the controller by feeding data to the motion card.

Interface to ARCS in C++

What was involved?

Reviewing the ARCS interface documentation and creating C++ code capable of sending data to the ARCS card.

What was completed?

The ACRS board included some sample C++ code that was implemented and tested to be sure that communications were established between the computer and controller. The team was successfully able to talk to the card and send and receive position information.

What was learned?

Interface to the ACRS motion card using C++. Recode the vision interface and computer vision algorithms using C++.

Interface to the Vision Card and Edge Detector Code in C++

What was involved?

This part of the project involved porting the VB code for the MATROX vision card and Edge Detector Algorithm from VB to C++.

What was completed?

The VB code was translated to C++ and the same testing was completed. Although C++ did not allow for as nice as an interface as VB the programming language transfer was successful and the team managed to put three of the major computer vision sections into

What was learned?

All three sections of the computer vision module can be successfully implemented using C++. The performance of C++ is slightly high than VB and this was important for tracking performance.
Kalman Filter Detector Design in VB and C++

What was involved?

Implementing the mathematical expression for the Kalman Filter into VB and then implementing the same code in C++ as C++ was found to be the programming language necessary for implementation.

What was completed?

The Kalman Filter was coded in VB and then recoded in C++. A problem was found regarding the implementation of a Kalman filter using a movable reference point (the camera’s field of view) because the Kalman Filter algorithm assumes a fixed reference point. This problem, in addition to the time left for the demonstration of the systems led the team to decide against pursuing implementation of the predictive model.

What was learned?

The group learned that changes will happen regardless of the schedule that is created. There will be problems that arise and as a team you have to adapt to these changes to take care of the top priorities, which for this project involved the realization of target tracking using computer vision.
3. Design Details

3.1 Control

3.1.1 Linear Modeling

As was stated before, our goal tracking speed was chosen to be 1 Ft/sec, which translates to Pan angular velocity of 0.1 radians/s or 5.7 degrees/s. All motor parameters were calculated and loaded into the MATLAB workspace via pantiltinitdesign.m (See Appendix), and pantilt_lin.m was executed to determine the open loop transfer function of the plant about the operating point of (0.1,0.1).

\[
G_{11} = \frac{0.001284s^3 + 0.002046s^3 + 239.1s^2 + 307.4s - 1370}{s^4 + 1.593s^3 - 5.337s^2 - 1.76s}
\]  

(1)

\[
G_{12} = \frac{-3.725e - 16s - 1.246e - 31}{s^3 + 1.593s^2 - 5.337s - 1.76}
\]  

(2)

\[
G_{21} = \frac{-3.725e - 16s^2 - 1.715e - 31s - 4.982e - 31}{s^4 + 1.593s^3 + 5.337s^2 - 1.76s}
\]  

(3)

\[
G_{22} = \frac{0.001398s^3 + 0.002227s^2 + 919.7s + 282.4}{s^3 + 1.593s^2 - 5.337s - 1.76}
\]  

(4)

The transfer functions for \( G_{12} \) and \( G_{21} \) are negligible meaning that the system is linear for the pan and tilt stages. The gains from the project proposal were used as an initial estimate for feedback. Kp1 = -1.60, Kd1 = -0.10, Kp2 = -2.00 and Kd2 = -0.03. Integral gain was added to both stages. Ki1 = -0.5 and Ki2 = -0.5. A washout filter was included to change the system from improper to proper. The gain K in Laplace form was given by

\[
K(s) = K_p + \frac{K_I}{s} + \frac{K_D}{p+1}
\]  

(5)

Using MATLAB’s SISO Design Tool the linear system was optimized for both stages.
The root locus of pan axis is shown below:

![Root Locus of the Pan Axis](image)

**Figure 3: Root Locus of the Pan Axis**

The plant of this pan axis is given by:

\[ G(s) = \frac{0.0012844(s^2 + 0.3071s + 1.862 \times 10^5)}{s(s + 0.3071)} \]  \hspace{1cm} (6)

The closed loop poles of the compensator were placed at:

\[ C(s) = \frac{17.0453(s + 15.51)(s + 10.04)}{s(s + 91.94)} \]  \hspace{1cm} (7)

The step response of the pan and the required torque is shown in the figure below:
Figure 4: Position and Torque Step Response for Pan Axis

Note that the optimal step response for the system is 0.1 radians. Therefore the above figure must be scaled down by a factor of 0.1. From the above figure it can be seen that neither the settling time nor the overshoot requirement is met for this controller. A key factor in the system design was the maximum peak torque output of 1.2 Nm. In light of torque restrictions, the percent overshoot was sacrificed for settling time. The required settling time of the pan axis was changed from 0.1 s to 0.2 s, because the team believed that having the camera settled by a short but specific time was more critical than having little or no overshoot. Experiments were performed on more aggressive controllers that far exceed the maximum peak torque output, that in simulation resulted in achieving both the percent overshoot and the settling time (1% and 0.1 s, respectively), produced high frequency chattering and did not move the axis.

The linearization and rttool method was applied to tilt axis in the same fashion as pan axis.
Figure 5: Root Locus of the Tilt Axis

The plant of this tilt axis is given by:

\[ G(s) = \frac{0.001398(s^3 + 1.286s + 6.579 \times 10^5)}{(s + 3.122)(s - 1.836)} \]  \hspace{1cm} (8)

The closed loop poles of the compensator were placed at:

\[ C(s) = \frac{15.0156(s + 23.95)(s + 13.24)}{s(s + 136.4)} \]  \hspace{1cm} (9)

The step response of the pan and the required torque is shown in the figure below:
As can be seen in the figure above the percent overshoot was sacrificed for the required settling time, but the torque requirement of 1.2 Nm is met so it was believed that the performance in Figure 6 could be met. This turned out to be not true as will be shown in section 3.1.5.

3.1.2 Friction Identification

Using MATLAB code (autoindent.m), provided by course TA Ben Potsaid, friction identification was completed on both the pan and the tilt axis. A one-degree of freedom model for coulomb and viscous friction was assumed. Coulomb friction is friction that is encountered before the motor turns the drive shaft and is typically referred to as ‘stiction’. Viscous friction is the friction encountered when the axis is moving. It is typically less then coulomb friction.

For a constant torque, $\tau$, the model of friction becomes:

$$F_v\dot{\theta} + F_c \text{sgn} \tau = \tau \quad (10)$$
In this equation (equation 10) $F_v$ represents the viscous friction and $F_c$ represents the coulomb friction.

A range of torques was selected for the identification, taking into consideration the total gear ratio and motor torque constant. A torque within the given range would be commanded to the motor and position values would be logged. The position data would be numerically differentiated to provide velocity data. By plotting the torque vs. velocity both the viscous and the coulomb friction could be found. The coulomb friction is the maximum applied torque commanded that does not move the shaft, and viscous friction is a function of velocities.

<table>
<thead>
<tr>
<th>Axis</th>
<th>Direction</th>
<th>Coulomb Friction</th>
<th>Viscous Friction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pan</td>
<td>Positive</td>
<td>0.09</td>
<td>0.0010219</td>
</tr>
<tr>
<td></td>
<td>Negative</td>
<td>-0.07</td>
<td>-0.001547</td>
</tr>
<tr>
<td>Tilt</td>
<td>Positive</td>
<td>0.10</td>
<td>0.001461</td>
</tr>
<tr>
<td></td>
<td>Negative</td>
<td>-0.09</td>
<td>-0.001335</td>
</tr>
</tbody>
</table>

Table 1: Experimental Friction Values

An approximate average was taken for these values for pan: 0.085 and 0.0013 for the coulomb and viscous frictions respectively, and for the tilt: 0.095 and 0.0014 for the coulomb and viscous frictions respectively.

3.1.3 Simulations with Friction

The friction values were placed back into the non-linear simulation model. Several simulations were completed to compare the effect of coulomb friction on the system. The model was modified to not include coulomb friction, to include coulomb friction, and finally to compensate for the coulomb friction. All of the models are considered for a step response of 0.1 radians. These are shown in the figure below:
Figure 7: Non-Linear Friction Simulations of Pan

From the figure above it can be seen that when coulomb friction is not considered non-linear, the response is very similar to the linear response for the pan axis in Figure 4. This is to be expected because the non-linear system was linearized about the step that was used to create the response (0.1 radians). When the coulomb friction is added to the non-linear simulation it is expected that the step response overshoot would not be as high, because coulomb friction acts to damp out motion until the torque is greater then the coulomb friction. This can be seen in Figure 7. The final line on the graph is the model with coulomb friction compensation. It would be expected that with coulomb compensation that the non-linear system would behave more like the linear system in terms of step response. This was not quite the case, as the settling time did reach the expected value of 0.2 s, but the percent overshoot did reach the value of the non-linear simulation. This may indicate that either something is not complete with the plant or that friction was obtained incorrectly.

Similarly, and for completeness the effects of coulomb friction are shown in Figure 8 below.
3.1.4 Real-Time Controller Design

The linear controllers obtained in section 3.3.1 were discretized using MATLAB’s ‘c2d’ function. The continuous domain transfer functions from both the pan and tilt axes were converted to discrete time, with time steps of 0.001s.

The resultant z-domain transfer functions are listed below:

Pan:

\[ C(z) = \frac{17.0453(z - 0.9867)(z - 0.9888)}{(z-1)(z-0.9122)} \]  \hspace{1cm} (11)

Tilt:

\[ C(z) = \frac{15.0156(z - 0.9796)(z - 0.9854)}{(z-1)(z-0.8725)} \]  \hspace{1cm} (12)
Also, friction compensation for coulomb friction was added to both axes. The entire controller that was implemented on the ARCS board is shown in the two figures below:

Figure 9: Pan Axis ARCS model with controller and friction compensation
3.1.5 Controller Performance

The performance of the pan and tilt axes was analyzed. Using a step input of 0.1 radians, ARCS’s AIDE data logging feature, a graph of the position response was obtained and can be seen in Figure 11.

Although friction compensation was added to the system, the non-linear simulation and the response of the physical system do not agree well for the pan axis. There is a slight increase in overshoot, but more troublesome is the extremely long settling time that occurs over 0.7 s. This was unexpected from our linear simulations, which predicted a settling time of 0.2 s. The non-linear simulation with friction compensation indicates a step response different from the linear simulation as well as the system.

From the pan step response, it can be seen that the proportional gain of our controller is too high for our system. Also, since the system speed is slow, either the derivative or integral gain is too large.
Figure 11: Pan Simulation and System Comparison

Similar comments can be made about the tilt axis. (Figure 12) The overshoot is larger than expected at over 60%, and the settling time is undesirable at almost 0.6 s. Again, this result was unexpected from the linear simulations. The proportional gain was too high, and either the derivative or integral gain was also too high.
Overall our tuned linear simulations did not seem to provide the response required for our system.

### 3.1.6 PID Controller

In an attempt to understand the system better, a proportional-integral-derivative (PID) controller was designed for the pan axis. Using the data acquisition functionality of AIDE, a controller was designed that would meet the specifications that were set out at the onset of the project. The authors of this report wish to point out that this controller was not implemented in the Final Presentation, as a majority of time was spent integrating vision and motion software.

The 0.1 radian step response is plotted, along with the non-linear simulation, below in Figure 13.
The proportional, integral, and derivative values were 8.0, 6.0, and 0.3, respectively. As can be seen in the graph, the percent overshoot is low (1.24%), and settles to a steady state error of 0.001 radians in less than 0.1 s. While this controller does not satisfy the sequential and logical process of design, it does meet the specifications of the system, and is only included as demonstration of what our optimal controller should have resembled.

3.2 Vision

The feedback for this controls project was chosen to be a vision system such that it can identify and track a moving point. This system consists of four levels of communication between the camera, frame grabber, CPU and the ARCS card. This communication was done in C++ in which we controlled the camera and frame grabber such that it grabbed frames at a certain rate and stored these in certain memory locations to be worked on by the CPU. Once the CPU has processed these images we need to send the ARCS card, through the ISA bus, the radian movements needed to operate the control system.

The first task in identifying our point will be to write an edge detector to identify the edge points in the image. Assuming an ideal environment like the environment our project will be working in, the task of identifying the point is quite simple. Once identification is
complete we track this current point through a combination of the edge detector and an incremental step function.

### 3.2.1 Roberts Edge Detector

Edge Detection can be split up into three main components: noise smoothing, edge enhancement, and edge localization. For the project purposes the noise smoothing will not be necessary because the simplicity of the environment we will be working in. Even if a large amount of noise is present, the edges on our point will not be disrupted too much to make it difficult to detect. For edge enhancement the Edge Detector we will be using is known as the Roberts Edge Detector. The Algorithm for this edge detector was found in [1]. (Trucco and Verri Reference)

### 3.2.2 Edge Enhancement

The task of determining which pixels are edge pixels and which ones are not is done by through the Roberts Edge Detector. Essentially, this algorithm calculates the first order image gradient of each pixel and through a threshold function makes a decision as to which pixels are edges. If we let \( I(c,r) \) be the image intensity at pixel \((c,r)\) then we can approximate first order image gradient by

\[
\nabla I(c,r) = \begin{pmatrix}
\frac{\partial I(c,r)}{\partial c} \\
\frac{\partial I(c,r)}{\partial r}
\end{pmatrix}
\]

where

\[
\frac{\partial I(c,r)}{\partial c} = I(c+1,r) - I(c,r)
\]

\[
\frac{\partial I(c,r)}{\partial r} = I(c,r+1) - I(c,r)
\]

The gradient magnitude is defined as:

\[
g(c,r) = \sqrt{\left(\frac{\partial I(c,r)}{\partial c}\right)^2 + \left(\frac{\partial I(c,r)}{\partial r}\right)^2}
\]

Finally mark as edges all pixels such that \(G(i,j) > \tau\). This threshold is determined through a testing process to find the best result for the application. Below is an example of the Roberts edge detection algorithm implemented in the lab.
Figure 14: Edge Enhancement
Original Screen Grab (left) and Edge Detection Output (right)

As we can see the edge detector works quite well. Most significant lines in our image were accounted for but there is still some noise in our output that could potentially throw off our calculations.

Figure 15: Edge Enhancement – Poor Lighting
Original Screen Grab (left) and Edge Detection Output (right)

Above is an example of how bad lighting affects the output of our edge detector. The original screen grab is of the same scene as Figure 14, but as can be seen the lighting is much worse. At the output we get an exceptional amount of noise as well a loss in edge detection quality. This also can attribute error in our overall environment and could be a reason why we encountered a lot of error in the classroom.
The images above show the effects of the focus on the edge detector. In the image, Chad is in the focus of the camera while the background is not. Clearly there is not much discrepancy between the objects that are in focus and those that are not. This proves that our focus does not have that much influence on the effectiveness of our system.

3.2.3 Edge Localization

Once all of the edge pixels in our image have been identified we are left with the issue of finding which pixels are the edges of the point we are tracking and then finding the center of this point. The question of finding which pixels are edge pixels to our point is not a trivial task in general so assumptions must be made. Generally this would be a pattern recognition problem which would find this point through a combination of curve fitting and probabilities, but because of the environment we are working in we can cut short this process. Assuming there will be nominal noise in our system and that only the torque to be tracked will be in the camera view; we will assume that all the edges obtained from the algorithm are the edges of the point. Therefore the center of this point is simply the mean of the detected edge pixels.

\[
\text{mean}_x = \frac{\sum \text{edgepixels}_x(n)}{n} \quad \text{mean}_y = \frac{\sum \text{edgepixels}_y(n)}{n}
\]  

(16)

This is the x and y coordinates of the center of the point we are tracking. From this information we can determine the movement of the controls.

3.2.4 Incremental Step Function

The tracking of the moving point will be done with a combination of the Roberts Edge Detector and an Incremental Step Function. We will run the edge detector in a loop continually calculating the center of our point in the current frame. Previously we took
these coordinates and simply changed them into a radian movement and had the controller move in that direction. Unfortunately this made our system jittery because it continued to try to move to keep this point in the center of the camera’s frame. To make up for this we implemented an incremental step function to smooth out our trajectory. Essentially what this function does is take the coordinates of the current position of our point and decide the direction of this point in relation to the camera and moves a step of 0.05 radians in that direction. While it slows down the speed at which we can track, it makes the system overall more robust.

<table>
<thead>
<tr>
<th>(0,0)</th>
<th>(640,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+,-</td>
<td>--</td>
</tr>
<tr>
<td>(320,240)</td>
<td></td>
</tr>
<tr>
<td>+,+</td>
<td>-,+</td>
</tr>
<tr>
<td>(480,0)</td>
<td>(640,480)</td>
</tr>
</tbody>
</table>

**Figure 17: Partition of image into four quadrants**

The above figure shows how the point relates to the camera pixels. We separate the image into four quadrants. Depending on which quadrant the coordinates of the image resides, we move accordingly in the pan and tilt directions. For example if the coordinates of our point was found to be in the lower right quadrant then the system will move -0.05 radians in the pan direction and 0.05 radians in the tilt. This leads to much smoother point to point movements but slows down the speed at which we can track.
3.2.5 Vision System Flowchart

Figure 18: Vision System Flowchart
3.3 Camera Bracket

The camera mounting bracket, which was used to mount the vision camera to the pan-tilt device, went through several incarnations in the design phase before being constructed. The original camera was a simple web camera, which was round, so the thought was to have a suction cup to hold it in place without having to perform any destructive modifications to the camera itself. The suction cup would then be connected to a base-plate that could be mounted to the tilt shaft of the system.

Figure 19: Original Web Cam Mount

However, it quickly became clear that this camera would not suit the project needs, as it had to be used with specific software, and the resolution was too low. The next camera to be selected, and the one that made it into the final design, was little more than a narrow cylinder. The camera initially came with a bracket that fit around its circumference and had a bolt that could be tightened to hold the camera securely in place by pinching the sides of the bracket together. This bracket was constructed out of aluminum and the bracket portion was machined narrow enough to be flexible for this purpose. The bracket also had two screw holes below the camera holder that could be used to screw it to a flat surface. The initial concept was to design a lower half for this bracket, with matching
holes to screw it together, and a hole through the bottom half of the bracket through which would pass the tilt shaft. These holes would be split at the bottom, similar to the hole holding the camera, so that bolts could be used to tighten the bracket around the tilt shaft as well.

![Figure 20: Camera Mount](image)

However, two specific problems surfaced with this design as well. First of all, it seemed too much of a nuisance to machine a mating part for the existing bracket, as it would be just as easy to remake the part that holds the camera on a whole new bracket. The other problem was that using bolts in order to tighten made the bracket too wide, prohibiting it from rotating very far within the tilt stage of the system. This problem was solved by choosing to use set screws to secure the bracket to the shaft. These screws passed along a diameter to the shaft hole itself and could be tightened down onto the shaft. Holes were drilled and threaded for four set screws, two on either side of the bracket, in order to provide maximum security. An additional innovation was to machine a plastic spacer between the bracket and the camera in order to protect the housing of the camera when the bracket was tightened. The bracket was machined out of solid aluminum bar stock, while the spacer was custom machined on a lathe out of PVC round stock. Because all design was done in-house, there were no contracting costs, and the only costs incurred were for materials.
Figure 21: Camera Mount with Spacer
4. Design Verification

4.1 Experimental Set-up

The original experimental set-up called for the pan-tilt system to be placed 10ft. away from a screen, onto which would be projected the target, consisting of a black dot on a white field. The initial idea was to use a physical target cut out of construction paper, and operated by hand. However, due to the sensitivity of the vision system, and the fact that the system used edge detection, the conclusion was that it would be too difficult to move the target using an arm or narrow dowel, as the edge detector would likely pick this up and become confused. Additionally, unless the target were flush against the background screen, depending on the lighting angles in the room, the shape might cast a shadow, making it appear larger than it really is. This led to the second concept, which was to use a computer animation, in this case, generated in MICROSOFT POWER POINT. Trials using this method clearly showed that the edge detector could single out the target dot, even though the color of the dot was not true black, nor was the background entirely white, due to the fact that this was a projection. However, when combined with the pan-tilt system, overall performance was less than satisfactory. This was revealed to be caused by the overshoot in the control system and the overall noisiness/ clarity of the image. To resolve this problem, the dot was reduced in diameter and a marked improvement in tracking was observed. However, due to the fact that both the dot and the background were of equal lighting levels, another contrast issue, the system was sometimes slow to track and would eventually lose the target as the speed increased. To solve this problem, the final set-up incorporated a laser pointer that projected the image of a star onto a dark screen. Tracking performance improved dramatically, and this was the set-up described in the final presentation. Although improvement in the tracking system was observed, the system did not perform quite up to the criteria set forth in the initial proposal. The majority of these problems likely stem from the control aspect of the project, as the controller was revealed to not be as fine-tuned as once thought. The reasons for this are described in detail in Future Considerations, in terms of inconsistencies between the predicted or modeled inertias and the actual system.

However, this is not to say that aspects of the vision system didn’t play a role in generating problems within the system. Possible sources of this are the fact that pixel resolution was related to angular velocity as a linear transformation, when in reality, the pan and tilt stages are tracing out a sphere, where objects will get correspondingly closer and further away as the two stages rotate.

Another aspect of the experimental set-up that went through design revisions was the zero/starting point for the pan-tilt system and the vision system. Originally, this was going to be the exact center of the screen. Using this information and the distance of 10ft that the device was placed at, it was a matter of simple geometry to determine the maximum angles to the right and left, up and down, from the center that the device would have to move. Using a desired linear speed for the target, it was also possible to
determine the angular velocities necessary in order to track the object. The location of this orientation point was later moved to the left upper-most corner of the screen. The device was still placed in the center of the target screen in order to minimize the effects of the difference between the spherical environment and the linear approximation (i.e. minimize the angle of rotation to the left and to the right. For the vision system, the orientation point was now the upper left-hand corner, because it works on the principle of a matrix, moving from left to right and from top to bottom to capture individual pixel information for filtering and comparison.

4.2 Verification

The final verification of the system can be divided into two sections; the first dealt with verification of the vision system, and the second the control system. For verification of the vision system, a graphical-user-interface (GUI) was designed, first in Visual Basic, and later duplicated in C++, when it was revealed that Visual Basic would not work with the ARCS card. This GUI demonstrated the operation of the edge detector and its performance by taking pictures of stationary objects and showing the edges that developed. The GUI also displayed the pan-tilt system’s field of vision while the system was operating. If it was working properly, the moving target would remain relatively close to the center of the screen.

For verification of the mechanical system, three programs were developed. One was used to verify that each stage could trace a straight line corresponding to its axis of rotation (i.e. a horizontal line for the pan stage, and a vertical line for the tilt stage). This program worked by simply feeding a list of equally spaced radial coordinates into the controller and observing as the system carried out the desired motion. The second program was meant to trace out a circle. This was important to demonstrate that both stages could work in unison. This program worked in a similar manner to the line program, where coordinates were fed into the pan and tilt stage, this time using the equation for a circle in order to transform between stages, \( x^2+y^2=1 \). In reality, this resulted in the system tracing out an elliptical shape because of the equidistantly spaced points, but the intent remained the same, to observe that the pan and tilt stages could be implemented together. The last program was the most simplistic, but very useful, and consisted of a jog function that would respond to the directional arrow keys on the computer keyboard. This was used to observe the movement of the pan and tilt stages in increments of 0.05 radians, the resolution that would be used in the final design.

As outlined in previous reports, the goal of the system verification was to observe the performance of the two subsystems of the project individually in order to isolate any problems before the final implementation. This was done to limit the possibility of design problems being translated into the combined system, as it might be hard to pinpoint the source of error at that stage.
4.3 System Performance

<table>
<thead>
<tr>
<th>System Performance Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Initial Specifications</td>
</tr>
<tr>
<td>--------------------------------</td>
</tr>
<tr>
<td>Maximum Tracking Speed (ft/s)</td>
</tr>
<tr>
<td>Settling Time (s)</td>
</tr>
<tr>
<td>Percent Overshoot</td>
</tr>
</tbody>
</table>

Table 2: System Performance Comparison

As we can see, the final performance was poor in comparison to the specifications set at the beginning of the semester. The settling time and overshoot are results of the discrepancies between our linear simulation and the nonlinear system. The linear simulation could have been tuned better but this would have led to a worse performance in the non-linear system. Despite these discrepancies our maximum tracking speed is still respectable. We set a somewhat high expectation for the specifications at the beginning of the semester and later learned that this most likely would not work out due to the processing speed. Therefore we are satisfied with our final tracking speed which was the major specification of the project.
5. Problems Encountered

The major problem that we encountered in the project was the discrepancy between the real time system controller and the simulated controller tuning. When tuning the controller through the linear simulation and *rltool*, we found no relation between how the system reacted through simulation and reality. When we placed our poles and zeros in *rltool* in such a way that the simulation looked excellent (zero overshoot and settling time less than 1s), the real system behaved totally differently. The system acted very weak and drifted a lot. To overcome this we had to increase the torque to make the system stronger. However this gave us a 50% overshoot and hurt our settling time. We feel the problems either exists in the friction calculations or with the Inertia Matrix. A temporary solution would be to use a simple PID controller block in SIMULINK and change the gains until we receive the performance needed as seen in section 3.1.6.

Another issue we encountered during the design process was the transformations between pixels coordinates and radian movement of the pan-tilt system. For the model presented at the demonstration, our math relied heavily that the distance the point of interest was from the camera remained static. However this is not true. As the pan/tilt moves the distance changes as a factor of the angle movement of the system. Therefore if the pan/tilt is at any position other than the home position the mathematics of the system movement are incorrect. This can be fixed through simple trigonometry functions but we side-stepped this issue by implementing the Incremental Step Function.

The final problem we encountered was interfacing the different components (MATROX, ARCS, CPU, etc.) of the project together. Initially we wrote our vision software in Visual Basic because of the simple interface between the MATROX software and VB. This is why we have the Edge Detector and Kalman Filter code found in Appendix 6 are in Visual Basic. Unfortunately we had problems communicating with the ARCS card so we had to rewrite our program in C++. Unfortunately the communication between C++ and MATROX was no longer trivial and it took much help from Course TA, Ben Potsaid, to have this working properly. Overall this is the problem that hurt us the most time-wise. If we did not have these issues we would have had more time to put towards improving our system.
6. Cost and Schedule

6.1 Cost Analysis

The costs associated with the project can be divided into three separate categories: mechanical system materials, computer vision system materials and labor. Included below is a breakdown of each category and the costs associated with each section.

6.1.1 Pan-Tilt Mechanism

The cost breakdown of the individual items is given in the table below:

<table>
<thead>
<tr>
<th>Part</th>
<th>Cost</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pan Motor</td>
<td>$89.80</td>
<td>Pittman: GM8724S010</td>
</tr>
<tr>
<td>Tilt Motor</td>
<td>$89.80</td>
<td>Pittman: GM8724S010</td>
</tr>
<tr>
<td>Pan Pulleys</td>
<td>$4.35 (x2)</td>
<td>SDP/SI: A 6Z 6-34DF02506</td>
</tr>
<tr>
<td>Pan Timing Belt</td>
<td>$2.92</td>
<td>SDP/SI: A 6B 6-123025</td>
</tr>
<tr>
<td>Tilt Pulleys</td>
<td>$17.69 (x2)</td>
<td>SDP/SI: A 6A 6-96NF01812</td>
</tr>
<tr>
<td>Tilt Timing Belt</td>
<td>$2.92</td>
<td>SDP/SI: A 6B 6-123025</td>
</tr>
<tr>
<td>Camera Mount Bracket</td>
<td>~$10(materials)</td>
<td>N/A, in-house</td>
</tr>
</tbody>
</table>

Total $239.52

The total cost for the pan-tilt mechanism is $285.91, which is just within our budget. Unfortunately, choosing a smaller motor for the tilt stage didn’t reduce the cost, the 6000 series motor from Pittman is actually significantly more expensive. However, price wasn’t the only motivating factor behind this change; it was also chosen to reduce the overall mass of the tilt stage and to increase efficiency. A different pulley was also selected for the pan stage. It was decided that the previous selections, with a diameter of less than 0.5 inch would put too much tension on the timing belt and would stress the pan motor as it tries to move the tilt stage, it having a relatively high inertia. For means of comparison, it would be like trying to spin a mass between two fingers instead of using the wrist. Greater leverage is possible with the latter.

6.1.2 Computer Vision

The camera, frame grabber board, driver software, and required cables have been obtained from the course TA, Ben Potsaid, and course instructor, Professor Wen. The computer vision part of the project will not increase our overall cost. A custom bracket will be needed in order to mount the camera to the tilt stage. This part will be machined by team member Tim Bagnall, negating any contracting costs and incurring only a small cost in materials.
6.1.3 Labor Cost

Labor cost is computed as follows:
Four individuals working 17 hours/week on the project gives 68 hours/week. Billing rate for engineers working on company internal project is approximately $100/hour. This yields $6,800/week. Assuming start date and end date of Jan 13, 2003 and April 30, 2003, respectively, this yields 15 weeks of work. Therefore, the total labor cost is $102,000.

6.2 Schedule

Illustrated below is the timeline for the project.

<table>
<thead>
<tr>
<th>Task</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hook camera and operate</td>
<td>2/10/2003</td>
</tr>
<tr>
<td>Test camera with the room</td>
<td>2/17/2003</td>
</tr>
<tr>
<td>Create PPT animation</td>
<td>2/24/2003</td>
</tr>
<tr>
<td>Edge Detector</td>
<td>3/3/2003</td>
</tr>
<tr>
<td>Kalman Filter</td>
<td>3/10/2003</td>
</tr>
<tr>
<td>User Interface</td>
<td>3/17/2003</td>
</tr>
<tr>
<td>RS-232 communications</td>
<td>3/24/2003</td>
</tr>
<tr>
<td>Design Proposal</td>
<td>3/31/2003</td>
</tr>
<tr>
<td>Progress Report</td>
<td>4/7/2003</td>
</tr>
<tr>
<td>Final Report</td>
<td>4/14/2003</td>
</tr>
<tr>
<td>Final Design Review</td>
<td>4/21/2003</td>
</tr>
<tr>
<td>Project Demonstration</td>
<td>4/28/2003</td>
</tr>
<tr>
<td>Final Presentation</td>
<td></td>
</tr>
</tbody>
</table>

Black – Hardware
Green – Software
Blue – Testing
Red – Written Reports
Orange – Presentations

Figure 22: Gant Chart
7. Professional and Societal Considerations

Codes, standards, patents, copyright issues, safety, environmental concerns, ethical dilemmas or possible economic impact on different segments of society did not affect our design. The design as a whole is an academic exercise whose purpose is to use the fundamentals of controls engineering to design, develop and implement a pan and tilt system.

Although our design was primarily used as an academic exercise there are parts of it that have social considerations when paralleled as a product in the real world. Particularly, our design best resembles the ‘real world’ products of an automatic/remote motion tracking devices, surveillance devices and inspection machines (ex. robots in an assembly line). When looking at these real world products there are social factors that need to be considered. Such considerations include federal codes such as the US Government Privacy Act of 1974, which states (in short form) that all agencies (federal or private) cannot maintain and collect information on individuals without their consent. If our system were used for surveillance then they could not be used to gain illegal information on particular users. For example, if our pan tilt system was used for security surveillance then it could not be used to collect information on people’s movement about a building. Also if our system exploited a particular technology (ex. a specific facial recognition algorithm) then verification would need to be completed to ensure that it does not violate existent patents.

Of course both of these societal considerations fall under the category of ethical dilemmas. Our product should be made so that by design is original and does not violate existent copyright or patent laws. In addition, the design should be aimed at applications that are legal and ethical. The product should be designed and marketed so as to influence its use in an ethical and appropriate manner. For example if our design was for public surveillance then it should not be marketed nor designed as an item to snoop on unsuspecting users.
8. Future Modifications

8.1 Kalman Filter

The Kalman Filter is a method reducing the search area for the point of interest by estimating the potential location of the point. Theoretically, we would not need to use this and just search the entire image for the point through a correlation function. However, this process would be computationally expensive and therefore would not allow us to track in real time. Therefore, the Kalman Filter can be used to estimate the image point’s position, velocity and localized search area. Knowing the estimated position and search region we can center the search region about this location to find the actual point using a correlation function.

The current state of the point being tracked is

\[ s_t = \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \end{bmatrix} \quad (17) \]

which represents the point’s position and velocity. The state vector at time \( t+1 \) is related to the current state by

\[ s_{t+1} = \Phi s_t + w_t \quad (18) \]

where \( \Phi \) is the state transition matrix and \( w_t \) represents the state perturbation modeled as zero-mean, white, Gaussian random processes. By assuming the movement between frames is linear then

\[
\begin{align*}
x_{t+1} &= x_t + v_{x,t} \\
y_{t+1} &= y_t + v_{y,t} \\
v_{x,t+1} &= v_{x,t} \\
v_{y,t+1} &= v_{y,t}
\end{align*}
\]

(19)

Then for \( s_{t+1} = \Phi s_t \)

\[
\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v_{x,t+1} \\ v_{y,t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \end{bmatrix}
\]

(20)

The measurement model for the filter is

\[ z_t = H s_t + \mu_t \quad (21) \]
where $H$ relates the current state to the current measurement and $\mu_t$ represents the measurement uncertainty. For simplicity we set the uncertainty to zero and then

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$  \hspace{1cm} (22)

The first step of our Kalman Filter algorithm is to predict the next state and the covariance estimation.

$$s_{t+1} = \Phi s_t$$

$$\Sigma_{t+1} = \Phi \Sigma_t \Phi^T + Q$$  \hspace{1cm} (23)

where $Q$ is the system perturbation and $\Sigma_{t+1}$ and $\Sigma_t$ are the current error covariance and the estimated error covariance respectively.

The measurement of these image points are executed with a correlation based search algorithm in a given search window. The correlation method we used is the sum of squared differences method for a given window of $d = [d_d, d_s]^T$

$$c(d) = \sum_{k=-W}^{W} \sum_{l=-W}^{W} \Psi(I_x(i+k, j+l), I_y(i+k-d_d, j+l-d_s))$$  \hspace{1cm} (24)

The size of the correlation window is determined by finding the values $\sigma_x$ and $\sigma_y$ which are the eigenvalues of the first 2x2 sub matrix of $\Sigma_{t+1}$. The size of the search window is then set to be $3\sigma_x \times 3\sigma_y$.

Once we have found the point we need to update our values for our next iteration.

$$K_{t+1} = \frac{\Sigma_{t+1} H^T}{H \Sigma_{t+1} H^T + R}$$  \hspace{1cm} (25)

where $K_{t+1}$ is the Kalman gain which is a weighting factor to determine the contribution of measurement to the posterior state estimate.

$$s_{t+1} = s_{t+1} + K_{t+1} \left( z_{t+1} - HS_{t+1} \right)$$

$$\Sigma_{t+1} = (I - K_{t+1} H) \Sigma_{t+1}$$  \hspace{1cm} (26) \hspace{1cm} (27)

For this algorithm to begin, we need a process to initialize the values to begin the prediction process. The initial state vector can be specified as

$$x_0 = x_{t+1}$$

$$y_0 = y_{t+1}$$

$$v_{x,0} = x_{t+1} - x_i$$

$$v_{y,0} = y_{t+1} - y_i$$  \hspace{1cm} (28)

The initial covariance matrix will be

$$\Sigma_0 = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$  \hspace{1cm} (29)

The initial system and error covariance matrices, $Q$ and $R$, also need to be determined.
The Kalman Filter was something that we planned on implementing since the beginning of the semester, but due to time constraints we were unable to employ this method. The lack of a Kalman Filter forced us to resort to tracking through edge detection which is not meant to be used for tracking because of the error involved. With this adaptive algorithm we will be able to predict the point’s future position and therefore decrease the search time, have a more accurate representation for the location of the point, and increase the overall speed of the system. The code for the Kalman Filter can be found in Appendix 6., however we were unable to test and implement the code into the project.

**8.2 Pattern Recognition**

The location of the center of the point of interest was done with the assumption that we were working in an ideal environment. In the real world we cannot make such an assumption. General noise and error are a common occurrence which would make our calculations deviate from reality. Therefore future work can be done to implement a pattern recognition algorithm to extract the point of interest from the edge pixels detected. This is an estimation problem of fitting a curve about our edge pixels and throwing away bad data that will throw off the calculations. Pattern Recognition is not a trivial task and is the main reason why this was not implemented.

**8.3 General Improvements**

There are obviously many more things that can be done to our project to make it operate more accurately or add different features that would make the system operate in a real world environment. The first and most prevalent is a more finely tuned controller. We need to work out the bugs between our linear simulations and non-linear system so that our tuning of the linear simulation will in turn tune the overall system. Next we could improve the system by adding in some trajectory generation. This should reduce most of the jitter that is attributed to the point to point motion. Another implementation would be to incorporate an adaptive parameter estimation process to find the mass, damping, and spring constant of the system. This will tell us how well our estimation of the Inertia Matrix is and if that plays a part in the errors we encountered. Finally by adding a second camera to the configuration we would be able to implement a stereoscopic system. This would allow us to determine depth information to track in 3D.
9. Conclusions

9.1 Results

Performance of the final design prior to the final presentation was deemed satisfactory. However, due to the fact that not all design criteria were met, and from comments/suggestions made by the instructor it is recommended that continued work be performed in order to refine the vision processing and control systems for this pan-tilt device. Continuing research and suggestions for future improvements are described in greater detail in the Future Considerations section of this report, and includes improvements to both the control system and the vision system. This project succeeded in mating the concepts of computer vision with closed-loop control of a linear system. Overall, the final system exhibited a strong knowledge of the structural concepts of controls engineering and performed to a standard that demonstrated a good deal of effort was put into bringing these theories to fruition in a physical application.

9.2 Multidisciplinary Aspect of the Project

This design process was one that drew up on skills from a wide range of previous course material. Obviously, basic control theory was the starting point and emphasis of this course. Without basic knowledge of linear control theory the task ahead of us would have been even more difficult. The physical design of the pan-tilt system put statics and dynamics into a realistic context as the team derived equations of motion for this mechanical system. In addition, because control was handled through a computer interface, computer communication became one area of our education to revisit. This was especially true for our team seeing that data processing was of critical importance for vision tracking. This leads us to the important topic of computer vision whose purpose was to acquire and process image information for control of the system. Through utilizing computer vision computer science was one discipline that became important as the algorithms that computer vision theory used could not be implemented without computer algorithms that were coded in C++. In addition, to realize the vision code knowledge of the hardware interfaces had to be known.

This project drew from courses ranging in basic metal machining to computer vision. Although the emphasis of the design process was based on control engineering, a task such as this would not have been realizable without a solid foundation in the basics of general engineering.
10. Individual Team Member Contributions

Contributions are defined as individual responsibilities that each team member had in the development and design of the project. The contributions listed do not reflect the individual team member’s assistance in creating this final report. For individual contributions regarding the project report please see the footnotes appended to each page.

10.1 Tim Bagnull

- Design and construction of the camera mount
- Camera mount and pan-tilt system CAD drawings
- Mechanical system analysis utilizing SOLIDWORKS
- Collection and compilation of team member resumes and material datasheets

10.2 James Deloge

- Vision card interface programming
- ARCS card interface programming
- Computer vision image processing program
- Motion control program
- Visual basic image capture and processing program
- Control path testing program
- Testing and verification of pan-tilt system
- Controller tuning and debugging

10.3 Chad Helm

- ACRS card interface programming
- Motion control program
- Testing and verification of pan-tilt system
- Controller tuning and debugging

10.4 Matthew Sked

- Edge detection algorithm programming
- Kalman filter algorithm programming
- Vision card interface programming
- Testing and verification of pan-tilt system
- Controller tuning and debugging
11. Final Report Authorship

11.1 Tim Bagnull
- Design Verification
- Cost and Schedule
- SOLIDWORKS Drawings
- CAD Drawings
- Conclusion

11.2 James Deloge
- Initial System Specification
- Design Methodology
- Professional and Societal Considerations
- Report Assembly and Formatting

11.3 Chad Helm
- Abstract
- Introduction
- Control Design Details

11.4 Matthew Sked
- Vision Design Details
- Problems Encountered
- Future Modifications
- Report Assembly and Formatting
12. References


Appendix 1: Pan System Geometry

Figure 23: Pan System Geometry
Appendix 2: Tilt System Geometry

Figure 24: Tilt System Geometry
Appendix 3: Pan-Tilt Parts Datasheets

1. Timing Belts
2. Pan Pulleys
3. Tilt Pulleys
4. GM8724S010 Motor
5. PG6614 Motor

Please refer to the subfolder “Parts Specification Sheets” in the “Technical Documentation” folder.
Appendix 4: Group Member Resumes

1. Tim Bagnull
2. James Deloge
3. Chad Helm
4. Matt Sked

Please refer to the subfolder “Team Resumes” in the “Final Report” folder.
Appenidix 5: testpath.cpp

/* File Name: testpath.cpp
 * Function: Sends motion control coordinates to the pan-tilt system according to user
 * choice. The user may test the system using a line, circle or jog function.
 * Coded By: James Deloge, Chad Helm
 * ECSE 4962-Controls System Design
 */

/*C++ Include Files*/
//==================================================
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
//==================================================
/*ARCS Include Files*/
//==================================================
#include "arcsdll.h"
#include "arcserror.h"
//==================================================
/*Define keyboard key constants*/
//==================================================
#define KEY_UP 72
#define KEY_DOWN 80
#define KEY_LEFT 75
#define KEY_RIGHT 77
//==================================================
/*Define DELAY constant*/
//==================================================
#define DELAY 10000000
//==================================================
/* Procedure Name: get_menu_choice
 * Function: Prints a user interface menu and waits until the user has made a choice
 */
int get_menu_choice(void);

int main()
{
    /*Define local variables.*/
    //=====================================================================  
    /*ARCS variables.*/
    //=====================================================================  
    char command1[] = "command";
    char command2[] = "command2";
    char position1[] = "position_one";
    char position2[] = "position_two";
    //=====================================================================  
    int get_menu_choice(void);
}
/*Input data variables.*/
//==================================================
char linefile[] = "LIN.TXT";
char circlefile[] = "CIR.TXT";
char datafile[8];
FILE *fp;
//==================================================
int i, status;

double value1;
double value2;

int offset = 0;
int choice = 0;
float reset = 0.0; // For resetting the tracing path positions
float value_command1 = 0.0;
float value_command2 = 0.0;

char *modName[50];
char *symName[50];
int modLen,symLen;
int ch;

double databuffer[7000];

int period = 1;
int noPoints = 5000;
int channel = 0;
int numT = 0;
int numP = 0;
//==================================================
/*====================================================================
* Initialize the ARCS board.
/*====================================================================
status = ArcsInitializeLocalHost ();
if(status !=0) { printf("<<ARCS>> %s \n",errorMsg[status]); exit(1);}
status = ArcsLoad ("pantiltexp");
if(status !=0) { printf("<<ARCS>> %s \n",errorMsg[status]); exit(1);}
printf ("Available Modules:\n");
status = ArcsGetModuleNames(modName,&modLen);
if(status ==0) {
printf("<<ARCS>> The available Modules are as follows...\n");
for(i=0;i<modLen;i++)
{
    if (i%2 == 0) printf("\n ");
    { printf("%20s ",modName[i]);
    }
    printf("\n\n");
}
else
{
printf(" <<ARCS>> %s \n", errorMsg[status]);
exit(1);
}
status = ArcsGetSymbolNames(modName[0], symName, &symLen);
if(status == 0)
{
    printf(" <<ARCS>> The available Symbols in <%s> are as follows...\n", modName[0]);
    for(i = 0; i < symLen; i++)
    {
        if((i % 2) == 0) printf("\n ");
        printf("%20s", symName[i]);
    }
}
printf("\n\n");
else
{
    printf(" <<ARCS>> %s\n", errorMsg[status]);
    exit(1);
}
//====================================================================
/*Start the application now.*/
status = ArcsStart();
//====================================================================
if(status == 0)
{
    printf(" <<ARCS>> current application is started...\n");
    status = ArcsGetSymbolValue(position1, &offset, &value1);
    status = ArcsGetSymbolValue(position2, &offset, &value2);
    printf(" Current Position 1: %f | Current Position 2 : %f \n", value1, value2);
}
else
{
    printf(" <<ARCS>> %s\n", errorMsg[status]);
}
/*====================================================================
* Print the user menu and start the program
/*====================================================================
/*While the user has chosen quit stay in the main program loop.*/
while (choice != 4)
{
choice = get_menu_choice();
/*====================================================================
* Jog Function
/*====================================================================
if (choice == 1)
{
    // Set the pan and tilt positions, *** DON'T use '_' on symbol
    printf("\n\n\n <Up> : Increase, <Down> : Decrease, <h> : Home Position _,
<q>: quit !\n\n");
    do
    {
        ch = _getch();

        if(ch == KEY_UP)
{ value_command2 = value_command2 + 0.05f;
 status = ArcsSetSymbolValue(command2,&value_command2,&offset);
 }
if(ch == KEY_DOWN)
{
 value_command2 = value_command2 - 0.05f;
 status = ArcsSetSymbolValue(command2,&value_command2,&offset);
 }
if(ch == KEY_LEFT)
{
 value_command1 = value_command1 + 0.05f;
 status = ArcsSetSymbolValue(command1,&value_command1,&offset);
 }
if(ch == KEY_RIGHT)
{
 value_command1 = value_command1 - 0.05f;
 status = ArcsSetSymbolValue(command1,&value_command1,&offset);
 }
if(ch == KEY_UP || KEY_DOWN || KEY_LEFT || KEY_RIGHT)
{
 status = ArcsGetSymbolValue(position1,&offset,&value1);
 status = ArcsGetSymbolValue(position2,&offset,&value2);
 printf(" Current Position 1: %f | Current Position 2 : %f \n", value1, value2);
 }
if(ch == 'h')
{
 value_command1 = 0.00f;
 value_command2 = 0.00f;
 status = ArcsSetSymbolValue(command1,&value_command1,&offset);
 status = ArcsSetSymbolValue(command2,&value_command2,&offset);
 }
if(status !=0)
{
 printf(" <<ARCS>> %s\n",errorMsg[status]);
 }
}
while(ch !='q');
//ArcsStop();
printf("Program is Quitting.\n");
/*===============================================================
* Circle or Line Function
/*===============================================================
else if(choice == 2 || choice == 3)
{
 if (choice == 2)
 {
 /* Straight Line Path.*/
 strcpy(datafile, linefile);
 }
else
{
 /* Circle Path.*/
 strcpy(datafile, circlefile);
 }
if ( (fp = fopen(datafile, "r")) == NULL) {
    fprintf(stderr, "Error opening line file.");
    exit;
} else {
    status = ArcsSetSymbolValue(position1, &reset, &offset);
    status = ArcsSetSymbolValue(position2, &reset, &offset);

    // Trace out the path of a line or circle
    while (!feof(fp)) {
        if (choice == 2) {
            fscanf(fp,"%f", &value_command1);
            printf(" %f", value_command1);
            status = ArcsSetSymbolValue(command1, &value_command1, &offset);
        } else if (choice == 3) {
            fscanf(fp,"%f %f", &value_command1, &value_command2);
            status = ArcsSetSymbolValue(command1, &value_command1, &offset);
            status = ArcsSetSymbolValue(command2, &value_command2, &offset);
            printf(" %f1 %f2", value_command1, value_command2);
        }
        for (i = 0; i < DELAY; i++) {
            //Do nothing delay
        }
        if(status != 0) {
            printf(" <<ARCS>> %s \n", errorMsg[status]);
            exit(1);
        }
    }
    fclose(fp);
}
}
ArcsStop();
return 0;

/* Procedure Name: get_menu_choice */
/* Function: Prints a user interface menu and waits until the user has made a choice */
int get_menu_choice(void) {
    int selection = 0;
    do {
    
/* Print the user menu. */
printf("\n");
printf("1  -  Jog Axis\n");
printf("2  -  Straight Line Path\n");
printf("3  -  Draw Circle Path\n");
printf("4  -  Quit\n");
printf("Enter a selection: \n");

/* Retrieve and return the user selection if it is valid. */
scanf("%d", &selection);
while (selection < 1 || selection > 4);
return selection;
Appendix 6: Kalman Filter Algorithm in Visual Basic

'Function Name: KalmanFilter
'Function:
'Encoded by: Matt Sked
'Team 1
'ECSE 4962 – Controls System Engineering

Public Function KalmanFilter (Arg As dataType,.........) As dataType

Dim NextImage(0 To 639, 0 To 479) As Byte
Dim ImageArray(0 To 639, 0 To 479) As Byte
Dim NewImage(0 To 6, 0 To 6) As Byte
Dim Identity(0 To 3, 0 To 3) As Byte
Dim SearchArea() As Byte
Dim Phi As New Matrix
Dim H As New Matrix
Dim Sigma0 As New Matrix
Dim Q As New Matrix
Dim R As New Matrix
Dim S As New Matrix
Dim z As New Matrix
Dim Sproj As New Matrix
Dim temp As Integer
Dim i As Integer
Dim j As Integer
Dim Sigma1 As Matrix
Dim PhiT As Matrix
Dim k As Matrix
Dim HT As Matrix
Dim TempMatrix As Matrix
Dim TempMatrix1 As Matrix
Dim TempMatrix2 As Matrix
Dim TempMatrix3 As Matrix
Dim TempMatrix4 As Matrix

'Declare 2 matrix variables that will hold
'the real and imaginary parts of EigenValues
Dim Vr As New Matrix, Vi As New Matrix

'Declare 2 matrix variables that will hold
'the real and imaginary parts of EigenVectors
Dim Er As New Matrix, Ei As New Matrix

Dim SigmaX As Integer
Dim SigmaY As Integer
Dim WindowsizeX As Integer
Dim WindowsizeY As Integer
Dim k As Integer
Dim l As Integer
Dim Corr As Integer
Dim Corr2 As Integer
Dim i_next As Integer
Dim j_next As Integer
Dim blStop As Boolean

'Initialization

'Define Phi Matrix
Phi.Size 4, 4
Phi(0,0) = 1: Phi(0,1) = 0: Phi(0,2) = 1: Phi(0,3) = 0
Phi(1,0) = 0: Phi(1,1) = 1: Phi(1,2) = 0: Phi(1,3) = 1
Phi(2,0) = 0: Phi(2,1) = 0: Phi(2,2) = 1: Phi(2,3) = 0
Phi(3,0) = 0: Phi(3,1) = 0: Phi(3,2) = 0: Phi(3,3) = 1

'Define H Matrix
H.Size 2, 4
H(0,0) = 1: H(0,1) = 0: H(0,2) = 0: H(0,3) = 0
H(1,0) = 0: H(1,1) = 1: H(1,2) = 0: H(1,3) = 0

'Define Sigma0 Matrix
Sigma0.Size 4, 4
Sigma0(0,0) = 50: Sigma0(0,1) = 0: Sigma0(0,2) = 0: Sigma0(0,3) = 0
Sigma0(1,0) = 0: Sigma0(1,1) = 50: Sigma0(1,2) = 0: Sigma0(1,3) = 0
Sigma0(2,0) = 0: Sigma0(2,1) = 0: Sigma0(2,2) = 25: Sigma0(2,3) = 0
Sigma0(3,0) = 0: Sigma0(3,1) = 0: Sigma0(3,2) = 0: Sigma0(3,3) = 25

'Define Q Matrix
Q.Size 4, 4
Q(0,0) = 16: Q(0,1) = 0: Q(0,2) = 0: Q(0,3) = 0
Q(1,0) = 0: Q(1,1) = 16: Q(1,2) = 0: Q(1,3) = 0
Q(2,0) = 0: Q(2,1) = 0: Q(2,2) = 4: Q(2,3) = 0
Q(3,0) = 0: Q(3,1) = 0: Q(3,2) = 0: Q(3,3) = 4

'Define R Matrix
R.Size 2, 2
R(0,0) = 4: R(0,1) = 0:
R(1,0) = 0: R(1,1) = 4:

'Define Identity Matrix
Identity(0,0)= 1: Identity(0,1) = 0: Identity(0,2) = 0: Identity(0,3)=0
Identity(1,0)= 0: Identity(1,1) = 1: Identity(1,2) = 0: Identity(1,3)=0
Identity(2,0)= 0: Identity(2,1) = 0: Identity(2,2) = 1: Identity(2,3)=0
Identity(3,0)= 0: Identity(3,1) = 0: Identity(3,2) = 0: Identity(3,3)=1

'Start the main program here

'Grab an image from the camera
Digitizer1.Grab
Image1.Get (ImageArray) 'Grabbed image array data

'Begin Edge Detection Algorithm

'Sked 2000 Edge Detection Code
'Edge Detector for Control Systems Design Project
'The aim of this program is locate the point that we will be tracking 'using our tracking program.
'Matthew Sked
'Initialize variables
Counter = 0
i_mean = 0
j_mean = 0

'Begin processing data
For i = 0 To 638
    For j = 0 To 478
'-----------------------------------------------------------------------------------------------------
        TempArray1 = ImageArray(i + 1, j + 1)
        TempArray2 = ImageArray(i, j)
        I1(i, j) = TempArray1 – TempArray2

        TempArray3 = ImageArray(i, j + 1)
        TempArray4 = ImageArray(i + 1, j)
        I2(i, j) = TempArray3 – TempArray4

        G(i, j) = Sqr(I1(i, j) ^ 2 + I2(i, j) ^ 2)
'-----------------------------------------------------------------------------------------------------
        If G(i, j) > 10 Then
            If i < 12 Or i > 18 Then
                FinalImage(i, j) = 0
                i_mean = i_mean + i
                j_mean = j_mean + j
                Counter = Counter + 1
            Else
            End If
        End If
    End For
Next i

'Check against divide by zero
If Counter = 0 Then
    Counter = 1
End If

'Calculate center of object
i_mean2 = i_mean / Counter
j_mean2 = j_mean / Counter
FinalImage(i_mean2, j_mean2) = 0

'Print Cross Hairs
For i_mean = -50 To 50
    FinalImage(i_mean2 + i_mean, j_mean2) = 0
Next i_mean

For j_mean = -50 To 50
    FinalImage(i_mean2, j_mean2 + j_mean) = 0
Next j_mean
'Begin Kalman Filter Algorithm

'State Transition Matrix
S.Size 4, 1
S(0,0) = i_mean2
S(1,0) = j_mean2
S(2,0) = 3 'Assumed Pixel Velocity
S(3,0) = 3 'Assumed Pixel Velocity

'Start main program loop
Do While blStop = True
    Digitizer1.Grab
    Image1.Get (ImageArray) 'Grabbed image array data

    'Calculates Correlation Window
    For i = i_mean2 - 3 To i_mean2 + 3
        For j = j_mean2 - 3 To j_mean2 + 3
            NewImage(i-i_mean2+3,j-j_mean+3)=ImageArray(i,j)
        Next j
    Next i

    'Projected State Transition Matrix => Sproj = Phi * S
    Set Sproj = Phi.Times(S)

    'Error Covariance Matrix
    Sigma1.Size 4, 4
    Set Sigma1 = Phi.Times(Sigma0)
    Set PhiT = Sigma1.Transpose
    Set Sigma1 = Sigma1.Times(PhiT)
    Set Sigma1 = Sigma1.Plus(Q)

    'Kalman Filter Gain
    k.Size 4, 2
    TempMatrix.Size 2, 4
    TempMatrix1.Size 2, 2
    Set HT = H.Transpose
    Set k = Sigma1.Times(HT)
    Set TempMatrix = H.Times(Sigma1)
    Set TempMatrix1 = TempMatrix.Times(HT)
    Set TempMatrix1 = TempMatrix1.Plus(R)
    Set TempMatrix1 = TempMatrix1.Inverse
    Set k = k.Times(TempMatrix1)

    For i = 0 To 1
        For j = 0 To 1
            TempMatrix1(i, j) = Sigma1(i, j)
        Next j
    Next i

    'Compute EigenValues and EigenVectors
    TempMatrix1.Eigen Vr, Vi, Er, Ei
    SigmaX = Vr(0, 0)
SigmaY = Vr(1, 0)
WindowsizeX = Round(1 / 2 * (3 * SigmaX))
WindowsizeY = Round(1 / 2 * (3 * SigmaY))

' Correlation Function
Digitizer1.Grab
Image1.Get (NextImage)
ReDim SearchArea(0 To (2*WindowsizeX-1),0 To (2*WindowsizeY - 1))

' Create a Search Area from the next image
For i = i_mean2 - WindowsizeX To i_mean2 + WindowsizeX
   For j = j_mean2 - WindowsizeY To j_mean2 + WindowsizeY
      SearchArea = NextImage(i, j)
   Next j
Next i

Corr2 = 0
' Big Window loop
For i = 2 To WindowsizeX - 3
   For j = 2 To WindowsizeY - 3
      ' Interior Loop
      For k = 0 To 6
         For l = 0 To 6
            temp = NewImage(k,l)*SearchArea(k+i-2,l+j-2)
            Corr = Corr + temp
         Next l
      Next k
      If Corr > Corr2 Then
         Corr2 = Corr
         i_mean2 = i
         j_mean2 = j
      End If
   Next j
Next i

' Bring coordinates back original frame
i_mean2 = Round(i_mean2 + [(640 - WindowsizeX)/2])
j_mean2 = Round(j_mean2 + [(480 - WindowsizeY)/2])

z.Size 2, 1
z(0, 0) = i_mean2
z(1, 0) = j_mean2

' State Transition Matrix Update
TempMatrix2.Size 2, 1
TempMatrix3.Size 4, 1
Set TempMatrix2 = H.Times(S)
Set TempMatrix2 = z.Minus(TempMatrix2)
Set TempMatrix3 = k.Times(TempMatrix2)
S = S.Plus(TempMatrix3)

' Error Covariance Update
TempMatrix4.Size 4, 4
Set TempMatrix4 = k.Times(H)
Set TempMatrix4 = i.Minus(TempMatrix4)
Sigma0 = TempMatrix4.Times(Sigma1)
'LOOP ENDS HERE
Loop
End Function
'-------------------------------------------------------------------------------------
Public Function FV(PV As Variant, i As Variant, n As Variant) As Variant
'Formula to calculate Future Value (FV)
'PV denotes Present Value
FV = PV * (1 + i / 100) ^ n
End Function
'-------------------------------------------------------------------------------------
Private Sub Command1_Click()
End Sub
Private Sub Continuous_Click()
  Digitizer1.GrabContinuous
End Sub
Private Sub Edge_Click()
End Sub
Private Sub Form_Load()
End Sub
Private Sub Stop_Click()
  blStop = False
End Sub
Appendix 7: grabfiltermotion.cpp

/* File name: grabfiltermotion.cpp
* Function: This program displays a continuous grab from the camera and implements a edge detector to
* determine the center of the moving target. The system then sends motion information to ARCS to move
* the pan-tilt system in the direction of the target motion.
* Coded By: James Deloge, Chad Helm
* Team 1
* ECSE 4962 – Controls Systems Engineering
*/

/*/C++ include files.*/
 //=============================================================
#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
//=============================================================

/*/MATROX vision card include file.*/
 //=============================================================
#include "mil.h"
//=============================================================

/*/ACRS control card include files.*/
 //=============================================================
#include "arcsdll.h"
#include "arcserror.h"
//=============================================================

/*/Image size constants.*/
 //=============================================================
#define IMAGE_WIDTH 640
#define IMAGE_HEIGHT 480
 //=============================================================

/*/Global variables.*/
 //=============================================================
int ImageArray[640*480], I1[640*480], I2[640*480], G[640*480], FinalImage[640*480];
double timer;
//=============================================================

void main(void)
{
    /* Matrox image card identifiers.*/
    MIL_ID MilApplication,  /* Application identifier. */
    MilSystem,  /* System identifier. */
    MilDisplay,  /* Display identifier. */
    MilDigitizer,  /* Digitizer identifier. */
    MilImage,  /* Image buffer identifier. */
    MilImage1,  /* Image buffer identifier. */
/*Local image variables. */
unsigned char *imageDataPtr;
char pictureFileName[] = "EdgeDetect.bmp";
long imagePitch;

/* Local edge detection variables.*/
int Counter, i, j;
const double distance = 75, width = 17, height = 15;
float theta1, theta2, imagey, imagex;
int i_mean, j_mean, i_mean2, j_mean2, temparray1, temparray2, temparray3, temparray4;
int xposition, yposition;

/*Local ARCS card variables.*/
int k, status;
char command1[] = "command";
char command2[] = "command2";
char position1[] = "position_one";
char position2[] = "position_two";

double value1;
double value2;

double initial_x;
double initial_y;

int offset = 0;

float value_command1 = 0.0;
float value_command2 = 0.0;

char *modName[50];
char *symName[50];
int modLen, symLen;

/*Initialize the ARCS card communications.*/
status = ArcsInitializeLocalHost ();
if(status != 0) { printf("<<ARCS>> %s \n", errorMsg[status]); exit(1);}
status = ArcsLoad ("pantiltexp");
if(status != 0) { printf("<<ARCS>> %s \n", errorMsg[status]); exit(1);}

// Illustration of getting application modules & symbols
printf ("Available Modules: \n");
status = ArcsGetModuleNames(modName,&modLen);
if(status == 0)
{
printf("<<ARCS>> The available Modules are as follows...\n");
for(k=0;k<modLen;k++)
{
    if( k%2 == 0) printf("\n ");
printf("%-20s ",modName[k]);
}
}
printf("\n\n");
}
else
{
    printf(" <<ARCS>> %s \n",errorMsg[status]);
    exit(1);
}
status = ArcsGetSymbolNames(modName[0],symName,&symLen);
if(status==0)
{
    printf(" <<ARCS>> The available Symbols in <%s> are as follows...\n",modName[0]);
    for(k=0;k<symLen;k++)
    {
        if( (k%2) == 0) printf("\n ");
        printf("%-20s ",symName[k]);
    }
    printf("\n\n");
}
else
{
    printf(" <<ARCS>> %s\n",errorMsg[status]);
    exit(1);
}
status = ArcsStart();
if(status !=0) { printf(" <<ARCS>> %s \n",errorMsg[status]); exit(1);}

/* Allocate defaults. */
MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, &MilDigitizer, _
&MilImage);

/* When a key is pressed stop the program. */
printf("Program started.\n");
printf("press any key to stop grabbing.\n");

status = ArcsGetSymbolValue(position1,&offset,&initial_x);
status = ArcsGetSymbolValue(position2,&offset,&initial_y);

/*Initialize the angles*/
theta1 = 0.0f;
theta2 = 0.0f;

/*Start the main program loop here.*/
while (!kbhit())
{
    /* Grab continuously. */
    MdigControl(MilDigitizer,M_GRAB_MODE, M_ASYNCHRONOUS);
    MdigGrabContinuous(MilDigitizer, MilImage);

    /* Allocate 2D buffer */
    MbufAlloc2d(MilSystem,IMAGE_WIDTH,IMAGE_HEIGHT,8L+M_UNSIGNED _,
    M_IMAGE+M_GRAB,&MilImage1);
/* Grab an frame to process*/
MdigGrab(MilDigitizer,MilImage1);

/*Initialize the wait timer and wait for specified amount of time while coordinates are
* computed
*/
timer = 0.08;
MappTimer(M_TIMER_WAIT,&timer);

/* Stop continuous grab. */
MdigHalt(MilDigitizer);

/* Set the pointer to the image buffer contents */
MbufInquire(MilImage1, M_HOST_ADDRESS, &imageDataPtr);

/* Find the image pitch */
MbufInquire(MilImage1, M_PITCH, &imagePitch);

/* Modify the image buffer */
MbufControl(MilImage1, M_MODIFIED, M_DEFAULT);

/*Process the image data using the Robert’s Edge Detector.*/
//==============================================================
/* Initilize variables */
Counter = 0;
i_mean = 0;
j_mean = 0;
for (i = 0; i < 640; i++)
{
    for (j = 0; j < 480; j++)
    {
        temparray1 = (int)imageDataPtr[(i + 1)+((j + 1)*imagePitch)];
        temparray2 = (int)imageDataPtr[i+(j*imagePitch)];
        I1[i+(j*imagePitch)] = temparray1 – temparray2;
        temparray3 = (int)ImageArray[(i + 1)+(j*imagePitch)];
        temparray4 = (int)ImageArray[i+(j*imagePitch)];
        I2[i+(j*imagePitch)] = temparray3 – temparray4;
        G[i+(j*imagePitch)] = (int)(sqrt (I1[i+(j*imagePitch)] ^ 2 + _
            I2[i+(j*imagePitch)] ^ 2));
        //Threshold originally 1
        if ((G[i+(j*imagePitch)] > 3) && ((i < 12) || (i > 18)))
        {
            FinalImage[i+(j*imagePitch)] = 0;    //Black 0
            i_mean = i_mean + i;
            j_mean = j_mean + j;
            Counter++;
        }
        else
        {
            FinalImage[i+(j*imagePitch)] = 255;  //White 255
        }
    }
}
if (Counter == 0)
{
    Counter = 1;
}

i_mean2 = i_mean / Counter;
j_mean2 = j_mean / Counter;

for (i = -50; i <= 50; i++)
{
    FinalImage[(i_mean2+i)+(j_mean2*imagePitch)] = 0;
}
for (j = -50; j <= 50; j++)
{
    FinalImage[i_mean2+(j_mean2+j)*imagePitch] = 0;
}

if (i_mean2 < 320)
{
    theta1 = theta1 + 0.05f;
} else
{
    theta1 = theta1 - 0.05f;
}

if (j_mean2 < 240)
{
    theta2 = theta2 - 0.05f;
} else
{
    theta2 = theta2 + 0.05f;
}
printf("X Radian %f: Y Radian %f \n", theta1, theta2);
//========================================

/*Send motion information to the ARCS card.*/
status = ArcsSetSymbolValue(command1,&theta1,&offset);
if(status !=0) { printf(" <<ARCS>> %s \n",errorMsg[status]); exit(1);}  
status = ArcsSetSymbolValue(command2,&theta2,&offset);
if(status !=0) { printf(" <<ARCS>> %s \n",errorMsg[status]); exit(1);}  
status = ArcsGetSymbolValue(position1,&offset,&value1);
status = ArcsGetSymbolValue(position2,&offset,&value2);
printf (" Current Position 1: %f | Current Position 2 : %f \n", value1, value2);

/* Save an EdgeDetection image to the file */
MbufExport(pictureFileName,M_BMP,MilImage1);

/*Free the buffer.*/
MbufFree(MilImage1);

/*Program finished. Return to home position*/
value_command1 = 0.00f;
value_command2 = 0.00f;
status = ArcsSetSymbolValue(command1,&value_command1,&offset);
status = ArcsSetSymbolValue(command2,&value_command2,&offset);

/*Stop running the arcs card.*/
ArcsStop();

/* Stop continuous grab. */
MdigHalt(MilDigitizer);

/* Save an EdgeDetection image to the file */
MbufExport(pictureFileName,M_BMP,MilImage1);

/* Clear the buffers and release defaults*/

/* Free the buffer */
MbufFree(MilImage1);
MbufFree(MilImage);
MbufFree(MilDisplay);

/* Release defaults. */
MappFreeDefault(MilApplication, MilSystem, MilDisplay, MilDigitizer, MilImage1);
MappFreeDefault(MilApplication, MilSystem, MilDisplay, MilDigitizer, MilImage);  
//========================================
% Initialization file for Pantilt Final Design
% Modified by Chad Feb 19, 2003

Ts = 0.001; % Sampling time of controller running on ARCS
sampletime = 0.002; % this is the data sampling in simulink
% Define motor and amplifier parameters
% Assume Motor A

Kmtr = 4.36e-2; % Motor torque constant Nm/A
motor1_igr=6.3; % Motor1 internal gear ratio
motor1_nls = 75.4*motor1_igr; % Motor1 no load speed at rotor in Rad/s
motor1_st = 3.0e-1/motor1_igr; % Motor1 stall torque at rotor in Nm
% Assume Motor B
% Where is the Kmtr for motor B
motor2_igr = 6.3; % Motor2 internal gear ratio
motor2_nls = 75.4*motor2_igr; % Motor2 no load speed at rotor in Rad/s
motor2_st = 3.0e-1/motor2_igr; % Motor2 stall torque at rotor in Nm

theta1_start = 0.0;
theta2_start = 0.0;
thdot1_start = 0.0;
thdot2_start = 0.0;

% Define gravity constant
g = 9.81;% acceleration due to gravity in m/(s^2)

% Define Joint 1 Parameters
belt1_gr = 96/24; % belt gear ratio for joint 1
%belt1_gr = 1;
N1 = belt1_gr*motor1_igr; % total gearing for the mass matrix
Ngear1 = belt1_gr*motor1_igr; % total gear ratio of internal and belt reduction
Imj1 = 1.6e-6; % inertia of motor rotor in KG*m^2 as seen by motor reference
Im1 = 1.6e-6;
fv1 = 0.01; % viscous friction for joint in NmS/Rad as seen by encoder
fc1 = 0.001; % coulomb friction for joint in Nm as seen by encoder

% the friggin motor we couldnt get
% % Define Joint 2 Parameters
% %belt2_gr = 96/24; % belt gear ratio for joint 2
% belt2_gr = 2.7; % was one
% N2 = 2;
% Ngear2 = belt2_gr*motor2_igr; % total gear ratio of internal and belt reduction
% Imj2 = 7.98e-7; % inertia of motor rotor in KG*m^2 as seen by motor reference
% Im2 = 7.98e-7;
% fv2 = 0.01; % viscous friction for joint in NmS/Rad as seen by encoder
% fc2 = 0.001; % coulomb friction for joint in Nm as seen by encoder

% Define Joint 2 Parameters
belt2_gr = 96/24; % belt gear ratio for joint 2
%belt2_gr = 1; % was one
N2 = belt2_gr*motor2_igr;                  % total gearing for the mass matrix
Ngear2 = belt2_gr*motor2_igr;           % total gear ratio of internal and belt reduction
Imj2 = 1.6e-6;                    % inertia of motor rotor in KG*m^2 as seen by motor reference
Im2 = 1.6e-6;
fv2 = 0.01;                    % viscous friction for joint in NmS/Rad as seen by encoder
fc2 = 0.001;                    % coulomb friction for joint in Nm as seen by encoder
torque_sat = 0.3*belt1_gr;                  % for the nonlinear simulation tor saturation

% Define Link A parameters for Team 1 setup
[pA, massA, IA] = bodyawithmotor;
m_A = massA;      % mass of link A in Kg
mA = massA;
lc_a1 = pA(1);
lc_a2 = pA(2);
lc_a3 = pA(3);
l_A = 0.0;                                  % location of CG on link A in m
I11_A = IA(1,1);
I12_A = IA(1,2);
I13_A = IA(1,3);
I22_A = IA(2,2);
I23_A = IA(2,3);
I33_A = IA(3,3);                                 % inertia of link 1 about CG in Kg*m^2
I21_A=I12_A;
I31_A=I13_A;
I32_A=I23_A;

% Define Link B parameters for team 1 setup
[pB, massB, IB] = bodybwithcamera;
m_B = massB;      % mass of link B in Kg
mB = massB
lc_b1 = pB(1);
lc_b2 = pB(2);
lc_b3 = pB(3);
l_B = 0.0;                                   % location of CG on link B in m
I11_B = IB(1,1);
I12_B = IB(1,2);
I13_B = IB(1,3);
I22_B = IB(2,2);
I23_B = IB(2,3);
I33_B = IB(3,3);                                % inertia of link 1 about CG in Kg*m^2
I21_B=I12_B;
I31_B=I13_B;
I32_B=I23_B;

% Constants for realtime simulation
Kdac= 0.1

% Design the controller
% Kp1 = -1.45;
% Kd1 = -0.1;
% Kp2 = -1.65;
% Kd2 = -0.05;
% K = [ Kp1 0 Kd1 0; 0 Kp2 0 Kd2 ];
Appendix 9: bodyawithmotor.m

% Calculate the mass and inertia properties for body A
% Chad Helm Feb 5, 2003
function [p_smph, m_smph, I_smph ] = bodyawithmotor;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
% Define some key dimensions and properties of the individual bodies. %
%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rho_al = 2.768e3;  % density of aluminum (kg/m^3)
dia_hub = 0.0381;  % diameter of hub on timing pulley
dia_pulley = 0.0630;  % outside diameter of timing pulley
dia_hole = 0.0095;  % diameter of hole in timing pulley
thick_hub = 0.0095;  % thickness of hub on timing pulley
thick_pulley = 0.0095;  % thickness of hub on timing pulley
thick_hole = 0.0095*2;  % thickness of hole in timing pulley
d1_payload = 0.0762;  % dimension in b1 direction of rectangular payload
d2_payload = 0.0381;  % dimension in b2 direction of rectangular payload
d3_payload = 0.0095;  % dimension in b3 direction of rectangular payload
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate the position to the center of mass, the mass, and the inertia tensor for each body %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for the skeletal yoke, pan, assembly
Pan_mass = 0.2294;
Pan_I11 = 0.0007459;
Pan_I12 = 0.0;
Pan_I13 = 0.0;
Pan_I22 = 0.0004750;
Pan_I23 = -0.0001031;
Pan_I33 = 0.0002978
Pan_I21 = 0;
Pan_I32 = 0;
Pan_I31 = 0;
Pan_I = [Pan_I11, Pan_I12, Pan_I13; ...]
       [Pan_I21, Pan_I22, Pan_I23; ...]
       [Pan_I31, Pan_I32, Pan_I33]
Pan_p = [0.0125; -0.0981; 0.0590]
% for the motor (for GM8724S010)
p_motor = [-0.0238 0 -0.0619]';
m_motor = 0.231;
r_motor = 0.0149;
h_motor = 0.0834;
I_motor = [1/12*m_motor*h_motor^2+1/4*m_motor*r_motor^2 0 0; ...]
      [0 1/12*m_motor*h_motor^2+1/4*m_motor*r_motor^2 0; ...]
      [0 0 0.5*m_motor*r_motor^2];
% for the pulley taken from Ben's body B program
% for the hub of the pulley
% p_hub = [0 -0.0588 0];
% m_hub = rho_al*pi*(dia_hub/2)^2*thick_hub;
% I_hub = [1/12*m_hub*(3*(dia_hub/2)^2+thick_hub^2) 0 0; ... 
% 0 1/2*m_hub*(dia_hub/2)^2 0 ; ... 
% 0 0 1/12*m_hub*(3*(dia_hub/2)^2+thick_hub^2)];
% for the outside of the pulley
p_pulley = [0 0 -0.0981];
% m_pulley = rho_al*pi*(dia_pulley/2)^2*thick_pulley;
% I_pulley = [1/12*m_pulley*(3*(dia_pulley/2)^2+thick_pulley^2) 0 0; ... 
% 0 1/2*m_pulley*(dia_pulley/2)^2 0 ; ... 
% 0 0 1/12*m_pulley*(3*(dia_pulley/2)^2+thick_pulley^2)];
% for the hole of the pulley
p_hole = [0 0 -0.0981];
% m_hole = rho_al*pi*(dia_hole/2)^2*thick_hole; % Note that hole is negative mass
% I_hole = [1/12*m_hole*(3*(dia_hole/2)^2+thick_hole^2) 0 0; ... 
% 0 1/2*m_hole*(dia_hole/2)^2 0 ; ... 
% 0 0 1/12*m_hole*(3*(dia_hole/2)^2+thick_hole^2)];
% for the skeletal tilt assembly
p_skel = [0 0.003 0];
% m_skel = 0.07583;
% I_skel = [ 0.0001284 0.0 0.0; 0.0 1.470e-6 0.0; 0.0 0.0 0.0001284 ];
% for the hub of the pulley
p_hub = [0 -0.0588 0];
% m_hub = rho_al*pi*(dia_hub/2)^2*thick_hub;
% I_hub = [1/12*m_hub*(3*(dia_hub/2)^2+thick_hub^2) 0 0; ... 
% 0 1/2*m_hub*(dia_hub/2)^2 0 ; ... 
% 0 0 1/12*m_hub*(3*(dia_hub/2)^2+thick_hub^2)];
% for the outside of the pulley
p_pulley = [0 0 -0.0981];
% m_pulley = rho_al*pi*(dia_pulley/2)^2*thick_pulley;
% I_pulley = [1/12*m_pulley*(3*(dia_pulley/2)^2+thick_pulley^2) 0 0; ... 
% 0 1/2*m_pulley*(dia_pulley/2)^2 0 ; ... 
% 0 0 1/12*m_pulley*(3*(dia_pulley/2)^2+thick_pulley^2)];
% for the hole of the pulley
p_hole = [0 -(0.0588+0.0493)/2 0];
% m_hole = rho_al*pi*(dia_hole/2)^2*thick_hole; % Note that hole is negative mass
% I_hole = [1/12*m_hole*(3*(dia_hole/2)^2+thick_hole^2) 0 0; ... 
% 0 1/2*m_hole*(dia_hole/2)^2 0 ; ... 
% 0 0 1/12*m_hole*(3*(dia_hole/2)^2+thick_hole^2)];
% for the payload
p_payload = [0 0 0.00831];
% m_payload = rho_al*d1_payload*d2_payload*d3_payload;
% I_payload = [1/12*m_payload*(d2_payload^2+d3_payload^2) 0 0; ... 
% 0 1/12*m_payload*(d1_payload^2+d3_payload^2) 0 ; ... 
% 0 0 1/12*m_payload*(d1_payload^2+d2_payload^2)];
% for the motor (from Prof Wen's website)
% p_motor = [-0.06 0 -0.06];
% m_motor = 0.231
% r_motor = 0.0348/2
% h_motor = 0.0739
% I_motor = [1/12*m_motor*h_motor^2+1/4*m_motor*r_motor^2 0 0; ... 
% 0 1/12*m_motor*h_motor^2+1/4*m_motor*r_motor^2 0 ; ... 
% 0 0 0.5*m_motor*r_motor^2];
% for the camera (measured by Chad)
% p_camera = [0 0 0.01750/2];
% m_camera = 0.07268;
% r_camera = 0.01750/2;
% h_camera = 0.05262;
% I_camera = [1/12*m_camera*h_camera^2+1/4*m_camera*r_camera^2 0 0;
% 0 1/12*m_camera*h_camera^2+1/4*m_camera*r_camera^2 0;
% 0 0 0.5*m_camera*r_camera^2];

% Start off with the skeletal body and add the hub of the pulley to form a composite body
% with new mass center at p_sh, new mass of m_sh, and new inertia tensor about p_sh of I_sh.
% Note that the hole has been included as a negative mass.
% add to form skeletal-motor
[p_sm, m_sm, I_sm] = compositebodies(Pan_p, Pan_mass, Pan_I, p_motor, m_motor, I_motor);
% add to form skeletal-motor-pulley
[p_smp, m_smp, I_smp] = compositebodies(p_sm, m_sm, I_sm, p_pulley, m_pulley, I_pulley);
% add to form skeletal-motor-pulley-hole
[p_smph, m_smph, I_smph] = compositebodies(p_smp, m_smp, I_smp, p_hole, m_hole, I_hole);

% Display the complete composite mass and inertia properties.
% p_smph
m_smph
I_smph
Appendix 10: bodybwithcamera.m

% Calculate the mass and inertia properties for body B
% Chad Helm Feb 5, 2003
function [p_shphc, m_shphc, I_shphc] = bodybwithcamera
%clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
% Define some key dimensions and properties of the individual bodies. %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
rho_al = 2.768e3;       % density of aluminum (kg/m^3)
dia_hub = 0.0381;       % diameter of hub on timing pulley
dia_pulley = 0.0630;    % outside diameter of timing pulley
dia_hole = 0.0095;      % diameter of hole in timing pulley
thick_hub = 0.0095;     % thickness of hub on timing pulley
thick_pulley = 0.0095;  % thickness of hub on timing pulley
thick_hole = 0.0095*2;  % thickness of hole in timing pulley
d1_payload = 0.0762;    % dimension in b1 direction of rectangular payload
d2_payload = 0.0381;    % dimension in b2 direction of rectangular payload
d3_payload = 0.0095;    % dimension in b3 direction of rectangular payload
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate the position to the center of mass, the mass, and the inertia tensor for each body %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for the skeletal tilt assembly
p_skel = [0 0.003 0]';
m_skel = 0.07583;
I_skel = [ 0.0001284 0.0 0.0; 0.0 1.470e-6 0.0; 0.0 0.0 0.0001284 ];
% for the hub of the pulley
p_hub = [0 -0.0588 0]';
m_hub = rho_al*pi*(dia_hub/2)^2*thick_hub;
I_hub = [1/12*m_hub*(3*(dia_hub/2)^2+thick_hub^2) 0 0; ...
        0 1/2*m_hub*(dia_hub/2)^2 0 ; ...
        0 0 1/12*m_hub*(3*(dia_hub/2)^2+thick_hub^2)];
% for the outside of the pulley
p_pulley = [0 -0.0493 0]';
m_pulley = rho_al*pi*(dia_pulley/2)^2*thick_pulley;
I_pulley = [1/12*m_pulley*(3*(dia_pulley/2)^2+thick_pulley^2) 0 0; ...
           0 1/2*m_pulley*(dia_pulley/2)^2 0 ; ...
           0 0 1/12*m_pulley*(3*(dia_pulley/2)^2+thick_pulley^2)];
% for the hole of the pulley
p_hole = [0 -(0.0588+0.0493)/2 0]';
m_hole = rho_al*pi*(dia_hole/2)^2*thick_hole;  % Note that hole is negative mass
I_hole = [1/12*m_hole*(3*(dia_hole/2)^2+thick_hole^2) 0 0; ...
          0 1/2*m_hole*(dia_hole/2)^2 0 ; ...
          0 0 1/12*m_hole*(3*(dia_hole/2)^2+thick_hole^2)];

% for the camera (measured by Chad)
p_camera = [0 0 0.01750/2];
m_camera = 0.07268;
r_camera = 0.01750/2;
h_camera = 0.05262;
I_camera = [1/12*m_camera*h_camera^2+1/4*m_camera*r_camera^2 0 0;... 
0 1/12*m_camera*h_camera^2+1/4*m_camera*r_camera^2 0;... 
0 0 0.5*m_camera*r_camera^2];

% Start off with the skeletal body and add the hub of the pulley to form a composite body %
% with new mass center at p_sh, new mass of m_sh, and new inertia tensor about p_sh of I_sh. %
% Note that the hole has been included as a negative mass. %
% %
% add to form skeletal-hub
[p_sh, m_sh, I_sh] = compositebodies(p_skel, m_skel, I_skel, p_hub, m_hub, I_hub);
% add to form skeletal-hub-pulley
[p_shp, m_shp, I_shp] = compositebodies(p_sh, m_sh, I_sh, p_pulley, m_pulley, I_pulley);
% add to form skeletal-hub-pulley-hole
[p_shph, m_shph, I_shph] = compositebodies(p_shp, m_shp, I_shp, p_hole, m_hole, I_hole);
% add to form skeletal-hub-pulley-hole-camera (added by Chad)
[p_shphc, m_shphc, I_shphc] = compositebodies(p_shph, m_shph, I_shph, p_camera, m_camera, I_camera);

% Display the complete composite mass and inertia properties. %

p_shphc
m_shphc
I_shphc
Appendix 11: compositebodies.m

% COMPOSITEBODIES calculates the mass and inertia properties for a 2 body composite
% Ben Potsaid Jan 21, 2003
% [pc, mc, Ic] = compositebodies(p1, m1, I1, p2, m2, I2)
% % -- input parameters --
% p1 is the 3x1 position vector from a coordinate reference frame to the center of mass of body 1
% m1 is the mass of body 1
% I1 is the 3x3 inertia tensor for body 1 about the center of mass of body 1
% p2 is the 3x1 position vector from a coordinate reference frame to the center of mass of body 2
% m2 is the mass of body 2
% I2 is the 3x3 inertia tensor for body 2 about the center of mass of body 2
% % -- returns --
% pc is the position vector to the center of mass of the composite body
% mc is the mass of the composite body
% Ic is the 3x3 inertia tensor for the composite body about the center of mass of the composite body

function [pc, mc, Ic] = compositebodies(p1, m1, I1, p2, m2, I2)

% calculate the center of mass for the composite body
[pc, mc] = masscenter(p1, m1, p2, m2);
% perform parallel axis to compute inertia tensor for the composite body
[Ic] = parallelaxis(p1, m1, I1, p2, m2, I2, pc);
Appendix 12: gravitylin.m

% Gravity Torques for pan-tilt
%

function gradG=gravitylin(mA,IA,pA,Im1,N1,mB,IB,pB,Im2,N2,theta1,theta2)

gradG=zeros(2,2);
g=9.8;
IA11=IA(1,1);IA12=IA(1,2);IA13=IA(1,3);
IA22=IA(2,2);IA23=IA(2,3);IA33=IA(3,3);
IB11=IB(1,1);IB12=IB(1,2);IB13=IB(1,3);
IB22=IB(2,2);IB23=IB(2,3);IB33=IB(3,3);
lcA1=pA(1);lcA2=pA(2);lcA3=pA(3);
lcB1=pB(1);lcB2=pB(2);lcB3=pB(3);
s2=sin(theta2);c2=cos(theta2);
% G(1)=0;
gradG(1,1:2)=[0 0];
% G(2)=-mB*g*c2*lcB1-mB*g*s2*lcB3;
gradG(2,1)=0;
gradG(2,2)=-mB*g*s2*lcB1-mB*g*c2*lcB3;
Appendix 13: masscenter.m

% masscenter: find the center of mass for a composite body
% Ben Potsaid Jan 21, 2003
% MASSCENTER calculates the center of mass for a composite body
% [pc, mc] = masscenter(p1, m1, p2, m2)
%
%       -- input parameters --
% p1 is the 3x1 position vector from a coordinate reference frame to the center of mass of body 1
% m1 is the mass of body 1
% p2 is the 3x1 position vector from a coordinate reference frame to the center of mass of body 2
% m2 is the mass of body 2
%
%       -- returns --
% pc is the position vector to the center of mass of the composite body
% mc is the mass of the composite body
function [pc, mc] = masscenter(p1,m1,p2,m2);

x1 = p1(1); y1 = p1(2); z1 = p1(3);
x2 = p2(1); y2 = p2(2); z2 = p2(3);
mc = m1+m2;
xc = (m1*x1 + m2*x2)/mc;
yc = (m1*y1 + m2*y2)/mc;
zc = (m1*z1 + m2*z2)/mc;
pc = [xc;yc;zc];
Appendix 14: massmatrix.m

% mass matrix for pan-tilt
%

function M=massmatrix(mA,IA,pA,Im1,N1,mB,IB,pB,Im2,N2,theta1,theta2)

    M=zeros(2,2);
    % run pantilt to obtain symbolic form for M first
    % the use ccode function on M(1,1), M(1,2), M(2,2)
    % and modify terms to match input
    IA11=IA(1,1);IA12=IA(1,2);IA13=IA(1,3);  
    IA22=IA(2,2);IA23=IA(2,3);IA33=IA(3,3);  
    IB11=IB(1,1);IB12=IB(1,2);IB13=IB(1,3);  
    IB22=IB(2,2);IB23=IB(2,3);IB33=IB(3,3);  
    lcA1=pA(1);lcA2=pA(2);lcA3=pA(3);        
    lcB1=pB(1);lcB2=pB(2);lcB3=pB(3);        
    s2=sin(theta2);c2=cos(theta2);  
    M(1,1)=IA33+mA*lcA1*lcA1+mA*lcA2*lcA2+Im1*N1*N1+IB11...  
        -IB11*c2*c2+mB*lcB2*lcB2+mB*lcB3*lcB3...  
        -mB*lcB3*lcB3*c2*c2-2.0*s2*c2*IB13+...  
        2.0*s2*c2*mB*lcB1*lcB3+c2*c2*IB33+e2*c2*mB*lcB1*lcB1;  
    M(1,2)=-s2*IB12+s2*mB*lcB1*lcB2+c2*IB23+c2*mB*lcB2*lcB3;  
    M(2,2)=IB22+mB*lcB1*lcB1+mB*lcB3*lcB3+Im2*N2*N2;    
    M(2,1)=M(1,2);
Appendix 14: pantiltlin.m

% linearized pan-tilt system about theta=0 thetadot =0
%
close all;
clear all;
clc;
% setup pan-tilt parameters
pantiltinitdesign;

% gravity
g = 9.807;

% get the linearized mass matrix
theta1d=0;theta2d=0;
M = massmatrix(mA,IA,pA,Im1,N1,mB,IB,pB,Im2,N2,theta1d,theta2d);

% assume no friction for now
d1=0.01; % viscous friction for pan
d1 = fv1
d2=0.01; % viscous friction for tilt
d2 = fv2
D = diag([d1 d2]);

% get the linearized gravity term
gradG=gravitylin(mA,IA,pA,Im1,N1,mB,IB,pB,Im2,N2,theta1d,theta2d);

% define state space matrices
A = [zeros(2,2) eye(2,2); -inv(M)*gradG -inv(M)*D];
B = [zeros(2,2) ; inv(M)];
C = [eye(2,2), zeros(2,2)];
% D = zeros(2,2);
G = ss(A,B,C,D);
disp('*** G ***');
disp(['Open loop transfer function linearized about (',num2str([theta1d theta2d]),')']);
tf(G)

G11=tf(G(1,1)); % pan characterization
G22=tf(G(2,2)); % tilt cahr
G11 = minreal(G11)
G22 = minreal(G22)
%
% controller parameters
%
% kp1=.1;kd1=.05;ki1=.1;
% kp2=.1;kd2=.05;ki2=.1;
kp1=1.6;kd1=0.1;ki1=.5;
kp2=2;kd2=.03;ki2=.5;
p=100;
K1=k_p1+kd1*tf([1 0],[1/p 1])+ki1*tf([0 1],[1 0]);
K2=k_p2+kd2*tf([1 0],[1/p 1])+ki2*tf([0 1],[1 0]);
K=[K1 0;0 K2];

% closed loop
Gcl11=feedback(G11*K1,1);
Gcl22=feedback(G22*K2,1);

% desired pan-tilt angles
theta1des=0.1;
theta2des=0.1;

% sampling period
ts = .005; % 200 Hz
close all
%step(Gcl22,0.2)
Appendix 15: paralleaxis.m

% PARALLELAXIS calculates the combine inertia tensor for a 2 body composite
% Ben Potsaid Jan 21, 2003
% [Ic] = paralleaxis(p1,m1,I1,p2,m2,I2,pc)
% 
% -- input parameters --
% p1 is the 3x1 position vector from a coordinate reference frame to the center of mass of body 1
% m1 is the mass of body 1
% I1 is the 3x3 inertia tensor for body 1 about the center of mass of body 1
% p2 is the 3x1 position vector from a coordinate reference frame to the center of mass of body 2
% m2 is the mass of body 2
% I2 is the 3x3 inertia tensor for body 2 about the center of mass of body 2
% pc is the 3x1 position vector to the center of mass of the composite body
% 
% -- returns --
% Ic is the 3x3 inertia tensor for the composite body about the center of mass of the composite body

function [Ic] = paralleaxis(p1,m1,I1,p2,m2,I2,pc)

% shift the first body to the new coordinate system at the composite center of mass
p1_c = pc-p1;
a = p1_c(1);
b = p1_c(2);
c = p1_c(3);
I1_c = I1 + m1*[ b^2+c^2 -a*b -a*c; -a*b c^2+a^2 -b*c; -a*c -b*c a^2+b^2];

% shift the second body to the new coordinate system at the composite center of mass
p2_c = pc-p2;
a = p2_c(1);
b = p2_c(2);
c = p2_c(3);
I2_c = I2 + m2*[ b^2+c^2 -a*b -a*c; -a*b c^2+a^2 -b*c; -a*c -b*c a^2+b^2];

Ic = I1_c + I2_c;
Appendix 16: sysmodcompare.m

%MAKE SURE THAT THE LOG FILE NAME IS THE SAME AS IN THIS FILE

load PIDPan.log
% find the time vector offset to compare the simulated and system response
clear newdata
i = 1
while PIDPan(i,2) < 0.002
    i = i + 1;
end

% Add the i offset to the time vector from simulink
% for j = 1 : i
%    subtime1(j,1) = (j-1)*0.001;
% end
%
% for k = 1 : 1000
%    subtime2(k,1) = time(k) + i*0.001;
% end
%
for j = i : i+1000
    newdata(j-i+1,1) = PIDPan(j,2);
end

% newtime = [subtime1;subtime2];
plot(time,states(:,1),time,newdata(:,1));
Appendix 17: tstats.m

function [Mo,tp,tr,ts,ess] = tstats(t,y,ref,yszone)
%TSTATS [Mo,tp,tr,ts,ess] = tstats(t,y,ref)
% Takes a time vector and a corresponding
% step-response vector and returns time-domain
% performance measures.
% Inputs: t - time vector
% y - step response corresponding to t
% ref - reference level at steady-state
% yzone - convergence zone
% Outputs: Mo - percent overshoot
% tp - time to peak
% tr - rise time (10% - 90%)
% ts - settling time (2%)
% ess - percent steady-state Error
%
% If the reference level, 'ref', is not specified, 1.0 is assumed.

if nargin < 3
    ref = 1;
    disp('reference value set = 1.0')
end
%
[maxy,itp] = max(y);
tp = t(itp);
Mo = 100*(maxy - ref)/ref;
if Mo < 0
    Mo = []; end
%
i10 = min(find(y>=0.1*ref));
i90 = min(find(y>=0.9*ref));
if i10 > 0 & i90 < length(y)
delt = t(2)-t(1);
t10 = t(i10) - delt*(y(i10)-0.10*ref)/(y(i10)-y(i10-1));
t90 = t(i90) - delt*(y(i90)-0.90*ref)/(y(i90)-y(i90-1));
tr = t90 - t10;
else
    tr = [];
end
%
is = max(find(abs(y - ref*ones(size(y)))/ref>0.02));
is = max(find(abs(y - ref*ones(size(y)))>yszone));
if is < length(y)
    ts = t(is + 1);
else
    ts = [];
end
%
%ess = abs(100*(y(length(y)) - ref)/ref);

% % % % % % % % % % % % % % % % % % end of tstats.m % % % % % % % % % % % % % %
Figure 25: SOLIDWORKS Drawing - Camera Mount Spacer
Figure 26: SOLIDWORKS Drawing - Camera Mounting Bracket
Figure 27: SOLIDWORKS Drawing - Camera
Figure 28: SOLIDWORKS Drawing - Camera with Mount
Figure 29: SOLIDWORKS Drawing - Vision Tracking Pan/Tilt System
Appendix 19: CAD Drawings

Figure 30: CAD Drawing – Camera Mount Buffer
Figure 31: CAD Drawing – Camera Mounting Bracket