STATE VARIABLE MODELING OF A VISUAL SERVO CONTROLLER FOR A MOBILE ROBOT

PREPARED BY
Jonathan Bruce

In partial fulfillment of the requirements for
MTRE 4000 – Dr. Ham

23 November 2010

SOUTHERN POLYTECHNIC STATE UNIVERSITY

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The project develops a vision-based mobile robotic system for wheeled mobile robots. In particular, this project focuses on developing a Full State Feedback Controller to control a wheeled mobile robot. First, a basic understanding of robotic kinematics and coordinate frame transformations is given for wheeled mobile robots. Then, based on Wang *et al.* a control law was developed to determine stability. Second, a Full State Feedback Controller is developed to control the motion of the robot based on visual error information. The simulation results show that the Full State Feedback Controller will control the robot quickly and accurately.

I. Introduction

Robotics has become a common field of study in science and engineering. In recent years, the field has seen a definitive shift in research and application. This shift can be seen in the demand for dynamic autonomous mobile robots which operate in natural or unstructured environments, such as planetary surfaces or in public areas. Some practical applications of these autonomous mobile robots are surveillance, search and rescue, handling hazardous material, or even mobile kiosks [1].

A large sector of research in this shift is the area of visual servo control or coined as visual servoing. This involves directly integrating visual data into a feedback control of the robot's position and orientation obtained from a CCD (Charge-Coupled Device) camera [2]. The use of visual servo control has been extensively researched and applied in structured industrial environments particularly in the automotive and electrical device manufacturing, but only recently has there been adequate technology and demand to extend visual servo control to a mobile robot base.

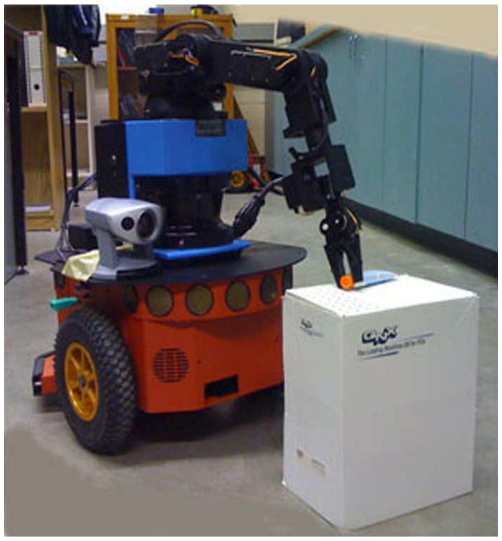


Figure 1: Pioneer P3DX w/Arm[1]

An emerging trend is to adapt the mobile robot base with an end effector for manipulation and grasping of various objects for short distance transportation or in areas where a fixed robotic arm would be impractical. An advantage of this is the ability for the visually controlled mobile robot to be more flexible to variations in applications. Also, terrain coverage and maneuverability out performs a fixed robotic manipulator [1].

II. **Problem Formulation**

A. Task Description

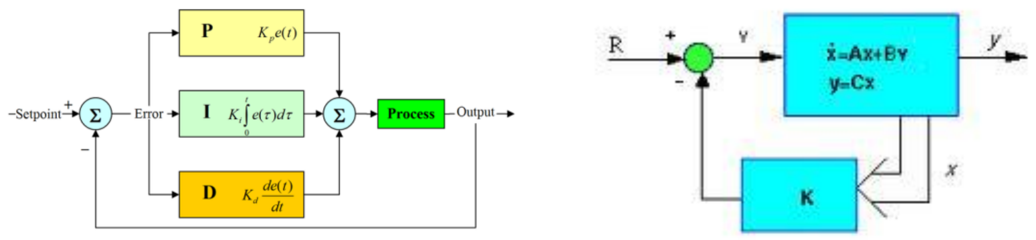
As stated above, there is a need for mobile robots with the capability to maneuver and manipulate objects autonomously. For this project, a task based on the paper written by Wang *et al.* [1] has been formulated. The objective of their paper was to develop a visual controller to be used in a multi-robot cooperative assembly project. A visual controller was developed and integrated with the capabilities of Pioneer DX3 mobile robot. This robot base was equipped a 5-DOF onboard manipulator arm, robot and arm shown in figure 1.

This nonholonomic vehicle, which is a vehicle having less controllable degrees of freedom then total degrees of freedom, has two wheel drive. Each wheel can be controlled independently, and they are equipped with a 500-pulse encoder to measure their relative positions and angular velocities. The 5-DOF arm and a color camera were mounted on the top of the robot base. These components were used as the vision controlled manipulation system. In the paper by Wang *et al.*, color-blob tracking software, ActivMedia Color Tracking System (ACTS), was utilized to track the target object which consisted of a toothpaste tube with a red cap. This scenario is the physical set-up that this project based on. Though, no physical experimentation was conducted, knowing the physical bounds in which the theoretical models are based helps to understand the entire system and its application.

B. Control Strategy

In contrast to the classical vision-based servo control scheme, the control approach of [1] decouples the control the mobile robot base from the control of the onboard 5-DOF end effector arm. This means the robot will utilize the visual information received from the CCD camera to position the base in close proximity to the object to allow the arm to grasp the object. Once the robotic base has moved to within a set distance of the object, the control of the base will cease and control will be switched to the grasping end effector arm. It was the intention of this project to develop a controller to manipulate both the robotic base and the end effector, but time limitations forced the exclusion of the end effector arm modeling and controller design.

The main difference between this project and [1] is the controller chosen to control the mobile robot. Wang *et al.* used a PID controller to control the movement of the robot, while this paper uses full state feedback control. PID or Proportional-Integration-Derivative controllers take into account three separate parameters: a proportional, an integral, and a derivative values. These values are used to try and eliminate errors in a feedback loop and adjust the process accordingly. Figure 2(a) shows a generic PID block diagram control loop. A full state feedback control is a control method used to place the closed-loop poles of a plant in user specified locations in the complex s-plane. This is desirable because the stability and characteristics of the system are directly related to the placement of a system's poles in the s-domain. Figure 2(b) shows a generic full state feedback controller block diagram control loop.



(a) (b)

Figure 2: (a) PID Block Diagram[5]; (b) Full State Feedback Diagram[6]

The physical model will utilize the CCD camera located on the top of the robot base to continuously capture and process live images of the target object. The current captured images are then compared to pre-recorded images of where the object should be located when the robot is within grasping distance. Using visual error between the current image position and the desired, the Image Based Visual Servoing (IBVS) controller will compute angular wheel velocities. These angular wheel velocities are what are sent down to the PID controller utilized by [1] and the full state feedback controller that was developed for this project. In both cases, the robot will translate and rotate continually till it reaches the desired position and ordination.

III. <u>Vision Based Robot Control</u>

We will now define the physical models that we will use to develop our controller and run our MATLAB/Simulink simulations.

A. Kinematic Model

In order to appropriately relate the physical robot's electromechanical system to the visual data received from our CCD camera, four sets of coordinate frames will need to be defined; the robot frame, the camera frame, the image plane frame, and the pixel coordinate frame. These four frames will be used to derive the IBVS controller used in [1] as well as develop the full state feedback controller. The first two frames to be defined are the robot frame and the camera frame. These two frames are by themselves easiest of the four frames to conceptualize. The robot coordinate frame is a Cartesian coordinate frame that is rigidly fixed to a point on the robot base. This point will move relative to the main world frame, but will help in locating the arbitrary robot origin as well as define the axes in which the robot can translate or rotate relative to its self. The camera coordinate frame is another Cartesian coordinate frame, though rigidly fixed to a point on the camera. This frame will aid in locating the direction the camera is facing relative to another coordinate frame as well as its pose (location in the world coordinate frame) and ordination relative to an arbitrary starting location.

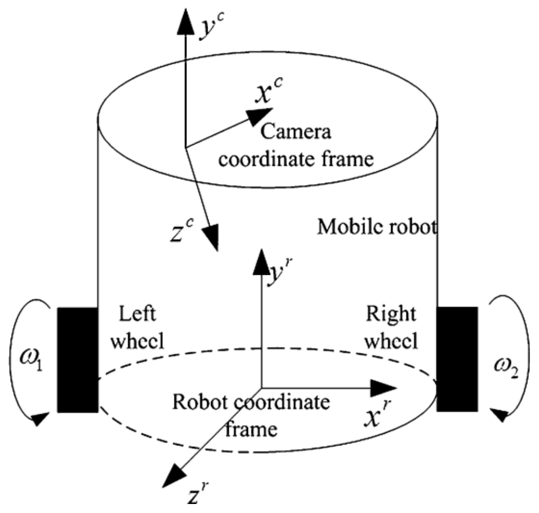
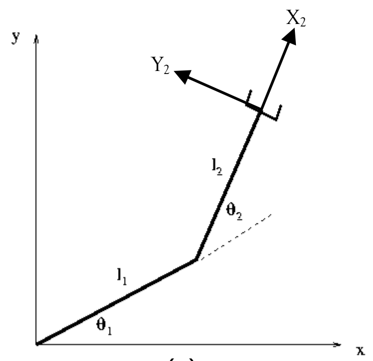


Figure 3: Relation of Robot Frame to Camera Frame

The coordinate transformation between the robot frame and the camera frame is given by

$$\mathbf{H}_c^r = \begin{bmatrix} \mathbf{R}_c^r & \mathbf{d}_c^r \\ 0 & 1 \end{bmatrix} \tag{1}$$

where Hrc is the homogeneous transformation from the camera frame to the robot frame. This means the transformation is can be either affine or perspective in nature. Now, Hrc is a matrix made of two smaller matrices, Rrc and drc. Rrc is the rotational matrix between the two coordinate frames. A rotational matrix relates how one coordinate plane is related to a base coordinate frame. A simple way to think about a rotational matrix is to think of what is called a two-link planar arm/robot, represented in figure 4. Now, a two-link planar arm is conceptually a robotic arm with two rotating joints that only has movement in a 2-D plane. As one can see the equations in figure 4(b) relate the end effectors rigid coordinate frame (X_2,Y_2) to both the main axes (X,Y).



(a)

$$\begin{gathered} X_2X_0 = \cos(\theta_1 + \theta_2) \\ X_2Y_0 = \sin\,(\theta_1 + \theta_2) \\ Y_2X_0 = -\sin\,(\theta_1 + \theta_2) \\ X_2Y_0 = \cos\,(\theta_1 + \theta_2) \end{gathered}$$

(b)

Figure 4: (a) Two-Link Planar Arm; (b) Two-Link Planar Rotation Relations

Thus, Rrc has the form

$$R_c^r = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The other matrix drc is a three dimensional vector that represents the position of camera origin to the robot frame. That vector has the form

$$d_c^r = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \tag{3}$$

Because of the rigid fixed frames the robot frame shares with the camera frame, their velocities can be expressed as six velocity vectors corresponding to the six degrees of freedom for a freely moving body. This relationship is derived in [5] and is obtained from (1).

$$\xi_{\mathrm{r}}^{\mathrm{r}} = \begin{bmatrix} 0 \\ 0 \\ v \\ 0 \\ \omega \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_c^r & \mathrm{s}(\mathbf{d}_c^r)\mathbf{R}_c^r \\ \mathbf{0}_{3\times 3} & \mathbf{R}_c^r \end{bmatrix} \xi_c^c = \mathbf{G}\xi_c^c \tag{4}$$

The matrix is formed is such a way as to give the user all the velocities of the camera that corresponds to the robot's translational and rotational velocity vectors. As one can see, the velocity vector for the robot only has two velocities. This is due to the robot's nonholonomonic design. The s(drc) signifies to create a skew symmetric 3x3 matrix from the position vector drc so that one might multiply it by Rrc. Then the last equation used to model the kinematics of the robot, is the equation which relates the robot's translational and angular velocities to the robot's wheel angular velocities. This matrix is given as

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} D/2 & D/2 \\ D/l & -D/l \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} \tag{5}$$

where D is the wheel diameter of the robot and l is the length between the two wheels. The vector with omega one and omega two are the individual wheel velocities for the robot.

B. Camera Model

This is where some abstract ideas about coordinate frames come into play. The next step in trying to understand our modeling is to relate the camera frame, the image plane frame, and the pixel coordinate frame. This is simple to comprehend when figure 5 is shown. In the figure, P is a point in the working frame or world frame. It has coordinates of $(x,y,z)^c$ relative to the camera frame. This means that from the camera's point of view, the object is located $(x,y,z)^c$ away from the origin of the camera.

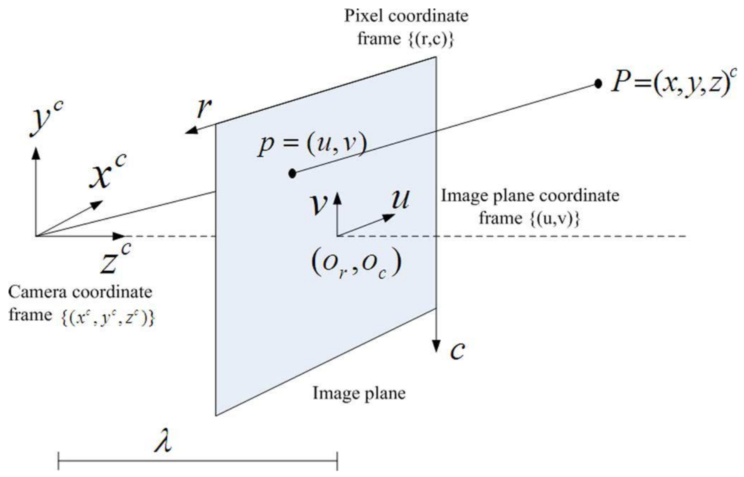

Pixel coordinate frame {(r,c)}
$P=(x,y,z)^c$
y^c
r
$p=(u,v)$
x^c
v
u
Image plane coordinate frame {(u,v)}
z^c
(o_r, o_c)
Camera coordinate frame $\{(x^c, y^c, z^c)\}$
c
Image plane
λ

Figure 5: Camera Frame, Image Plane Frame, and Pixel Coordinate Frame Relationship

Now, lower case p is the projection of capital P on to the image plane with coordinates (u,v). These coordinates are relative to the image coordinate frame. Now the image plane is directly related to the pixel coordinate frame. The coordinates (r,c) directly relate to the coordinates of p in the pixel coordinate frame. The distance from the origin of the camera frame to the image plane is denoted by λ, which is also known as the focal length of the camera. The points (o_r,o_c) is known as the principal point with respect to the pixel coordinate frame. The linear relationship between the image plane coordinates and the pixel coordinate frame coordinates are shown to be derived from [5] and are

$$\begin{cases} u = -s_x(r - o_r) \\ v = -s_y(c - o_c) \end{cases} \tag{6}$$

where sx and sy are the horizontal and vertical dimensions, respectively, of any given pixel. Another rather lengthy and mathematically taxing equation that has been derived in [5] is called the interaction matrix $\mathbf{L}$. It relates the camera velocity to the individual velocities of the image plane coordinates. This matrix is given below.

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \mathbf{L}\boldsymbol{\xi}_{\mathbf{c}}^{\mathbf{c}} = \begin{bmatrix} -\frac{\lambda}{z^c} & 0 & \frac{u}{z^c} & \frac{uv}{\lambda} & -\frac{\lambda^2 + u^2}{\lambda} & v \\ 0 & -\frac{\lambda}{z^c} & \frac{v}{z^c} & \frac{\lambda^2 + v^2}{\lambda} & -\frac{uv}{\lambda} & -u \end{bmatrix} \boldsymbol{\xi}_{\mathbf{c}}^{\mathbf{c}} \quad (7)$$

The depth value z^c can be determined by the onboard distance range sensor of the robot, usually laser or ultra sonic sensors.

C. Control Law

For this project, I am using the image-based eye-in-hand visual servo control law developed by Wang *et al*. The purpose of this is to understand how a control law is developed and used, as well as to allow for this project to focus on developing a full state feedback controller. The idea of this eye-in-hand IBVS controller will be to continuously adjust the wheel velocities, and therefore adjust the robot, to move the image coordinates of the tracked object to a desired position in the image plane. A desired position must then be defined as (u_d,v_d). We then get an error equation for our image plane coordinates

$$\mathbf{e} = \begin{bmatrix} u - u_d \\ v - v_d \end{bmatrix} \quad (8)$$

Since the image plane coordinates are directly proportional to coordinates in the pixel reference frame, e can be expressed as

$$\mathbf{e} = \begin{bmatrix} -s_x(r - r_d) \\ -s_y(c - c_d) \end{bmatrix} \quad (9)$$

Then, take the time derivative of this to get an error velocity equation. This equation will be very advantageous, because the image plane velocity vector is equal to a known relation which can be used to relate it back to the robot sytem. So, e dot can be represented as

$$\dot{\mathbf{e}} = \begin{bmatrix} \frac{d(u-u_d)}{dt} \\ \frac{d(v-v_d)}{dt} \end{bmatrix} = \begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} \tag{10}$$

Because the desired image positions are constants when the derivative of e is taken, the constants will be turned into zeros and we are left with just the time derivatives of our current position vector in the image plane. Through some mathematical manipulations, the image velocity error vector can be set equal to the velocities of the robot. First, replace the velocity vector (u dot, v dot)^T with the interaction matrix $\mathbf{L}$ defined as

$$\dot{\mathbf{e}} = \mathbf{L}\boldsymbol{\xi}_{\mathbf{c}}^{\mathbf{c}} \tag{11}$$

Next, take the relation equation between the robot velocity vector and the camera velocity vector and solve for the camera velocity. Then e dot equals the interaction matrix times the inverse of **G** times the robot velocity vector.

$$\dot{\mathbf{e}} = \mathbf{L}\mathbf{G}^{-1}\boldsymbol{\xi}_{\mathbf{r}}^{\mathbf{r}} \tag{12}$$

Now, since the robot velocity vector only has two non-zero components, we can create a new matrix **M** such that it comprises of the corresponding columns of **LG**$^{-1}$, those column vectors are the 3rd and 5th column vectors of **LG**$^{-1}$. The image velocity error vector then equals

$$\dot{\mathbf{e}} = \mathrm{M}\begin{bmatrix} v \\ \omega \end{bmatrix} \tag{13}$$

Where $(v,w)^T$ is the translational and rotational velocities of the robot base. Finally, the last kinematic model that related the robot base velocities to its wheels angular velocities can be substituted. Then some simple algebra will give us an equation relating image velocities to angular velocities of the wheels on the robot.

$$\dot{\mathbf{e}} = \mathbf{M} \begin{bmatrix} \frac{D}{2} & \frac{D}{2} \\ \frac{D}{l} & -\frac{D}{l} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} \tag{14}$$

$$\begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} \frac{D}{2} & \frac{D}{2} \\ \frac{D}{l} & -\frac{D}{l} \end{bmatrix}^{-1} \mathbf{M}^{-1}\dot{\mathbf{e}} \tag{15}$$

The next step that [1] took was to develop a controller based on the Lyapunov method. This assumes e dot equals a negative scalar gain time the position error. Replace the e dot in our last equation and we get a controller that [1] states will compute the desired angular velocities of the two wheels of the mobile robot from the image measurements. Also, [1] states the controller guarantees asymptotic stability of the closed-loop system. This means that for any positive k value, the controller will push the poles of the controller to the negative side of the s-complex plane.

$$\dot{\mathbf{e}} = -k\mathbf{e}. \tag{16}$$

$$\begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} \frac{D}{2} & \frac{D}{2} \\ \frac{D}{l} & -\frac{D}{l} \end{bmatrix}^{-1} \mathbf{M}^{-1}(-k\mathbf{e}) \tag{17}$$

$$\begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = -k \begin{bmatrix} \frac{D}{2} & \frac{D}{2} \\ \frac{D}{l} & -\frac{D}{l} \end{bmatrix}^{-1} \mathbf{M}^{-1} \begin{bmatrix} -s_x(r - r_d) \\ -s_y(c - c_d) \end{bmatrix} \tag{18}$$

IV. <u>Control Design</u>

A. State Variable Model

State variable modeling is a representation of physical system as a mathematical model. This model has a set of inputs, outputs, and state variables. State variables are chosen in such a way where they can be represented by first-order differential equations. The model is then set up as a linear system of equations with a solution vector corresponding to the time derivatives of the state variables in the system. The coefficients of the differential and algebraic equations that relate the state variables to its time derivative and output are written as matrices. The generalized from of a state variable model is given in the equation given below.

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{19}$$

In equation (19): **x** is your state variable, **u** is the system input, and **y** is the system output. The convenience of using state variable modeling over another method like frequency domain analysis or Laplace transforms, is that the state space model is invariant to multiple inputs, multiple outputs, and initial conditions. A minor drawback to state space modeling is that it can only handle linear systems.

B. State Variable Model for a Mobile Robot Vision Based Control

A state variable model of the mobile robot system is needed before we can develop a control scheme. The system will take wheel velocity values given it and in turn make the robot move, changing the current image in the camera image frame. Thus, our input vector will be angular wheel velocities and our state variables will be the error of our pixel coordinate frame coordinates. These relations are given as

$$\underline{u} = [\omega_1 \; \omega_2]^T \tag{20}$$

$$\underline{x} = [r \; c]^T \tag{21}$$

$$\underline{\dot{x}} = [\dot{r} \; \dot{c}]^T \tag{22}$$

With this in mind, equation (19) can be used directly as our state equation. Therefore, the next state of our system is only dependent on the input to the system. The output equation will be the state variable matrix. The matrix values for the state variable model are

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{23}$$

$$B = \boldsymbol{M} \begin{bmatrix} \frac{D}{2} & \frac{D}{2} \\ \frac{D}{l} & \frac{-D}{l} \end{bmatrix} \tag{24}$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{25}$$

C. Full State Feedback Control

Full state feedback, as described in the first section, is a control method used to place the closed-loop poles of a plant in user specified locations in the complex s-plane. The idea behind this method is that the state variables, which are the major representations of a system, can be manipulated to push the poles of the characteristic equations of the system to specific user defined poles. The first step developing a full state feedback controller is to set the input vector as

$$\underline{u} = -K\underline{x} \tag{26}$$

where **x** is the current state variable and K is a scalar gain constant. Place (26) back into the state equation, and get

$$\underline{\dot{x}} = (A - BK)\underline{x} \tag{27}$$

A typical method to solve for a K value involves using the characteristic equation for the state space equation, choose pole locations, and solve a polynomial equation with K values based on predetermined pole locations.

$$\det(\mathrm{sI} - \mathrm{A}) \Rightarrow \det[sI - (A - BK)] = \det(sI - A + BK) = 0 \tag{28}$$

The issue with the approach mentioned is that it only works for SISO (Single Input Single Output) systems. In the MIMO (Multiple Input Multiple Output) case, the K value becomes non-unique and it is not trivial to calculate since there are more K values then equations. Another method must be used in order to calculate the gain value.

The use of a Linear-Quadratic Regulator function is to determine a gain constant for a system. A LQR (Linear-Quadratic Regulator) function uses a mathematical algorithm that calculates a K which minimizes the cost based on the given cost factor and weighting factor [7]. The base formula used is

$$J = \frac{1}{2}x^T(t_1)F(t_1)x(t_1) + \int_{t_0}^{t_1} \left(x^TQx + u^TRu\right) dt \qquad (29)$$

For this project, the system is a MIMO. Therefore, a LQR calculation was required to compute the value of K. The LQR calculations were computed in MATLAB using the Control Systems Toolbox.

V. <u>Experimental Results</u>

A. MATLAB Model

MATLAB by MathWorks was used to model and develop the controller for the vision based controller. Utilizing the Control Systems Toolbox built into MATLAB, the creation of a state variable model was easy. Since MATLAB is a computational tool based on the assumption that all inputs are matrices, the state matrices were directly typed into MATLAB. Then the State-Space Model function in the Control Systems Toolbox was called to create a state space variable to hold the system matrices. Once the state variable model had been stored in MATLAB, another function called “lsim” was used to simulate the response of the continuous state variable system. An input value of one was utilized though out the simulation. The main MATLAB code used to compute all data and graphs can be found in Appendix A along with references to the specific code that created each MATLAB figure. Also, the values used for the system are given in Appendix A.

The initial step was to input the state matrices into MATLAB as a state-space model and simulate the uncontrolled response. As one can see in figure 6, the system is quite unstable.

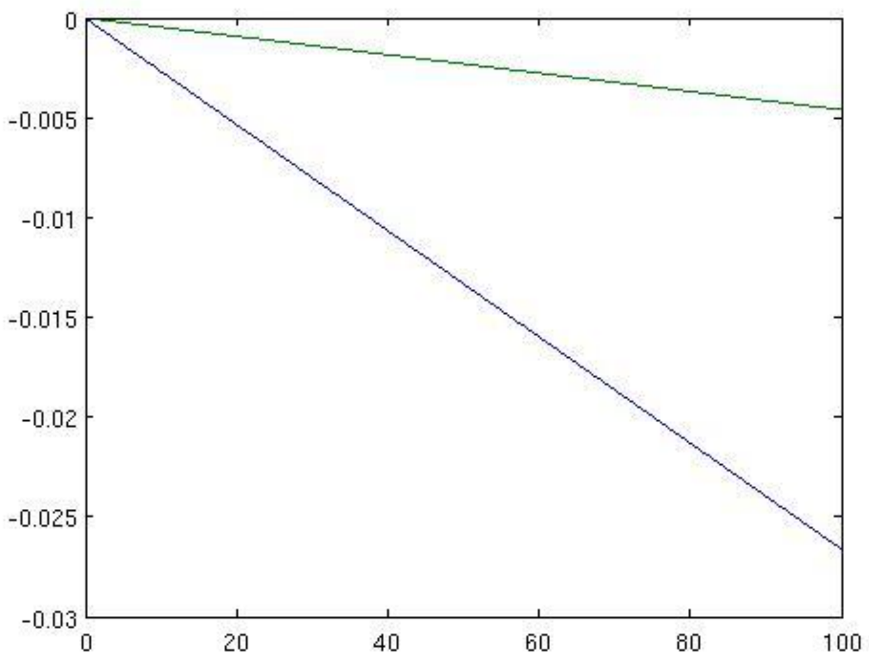


Figure 6: Uncontrolled System Response

For this reason, a constant gain was needed to be applied to the system to help stabilize the output response. As mentioned in the last section, a Linear-Quadratic Regular function was used to calculate the gain value. Based on the uncontrolled response and the system constant values, a large value for the cost variable was used. In order to calculate a generic cost value, one can use the formula given as

$$Q = C' * C \tag{30}$$

where C is the output matrix from my state variable model's output equation. This value gives a Q equal to the 2x2 identity matrix since C is a 2x2 identity matrix. The Q matrix is then multiplied by 500000 to compute a large cost value for the LQR function. This gives

$$Q = \begin{bmatrix} 500000 & 0 \\ 0 & 500000 \end{bmatrix} \tag{31}$$

Using this Q value along with a weighting value of one, a gain matrix of (32) was determined.

$$K_1 = \begin{bmatrix} -221.8223 & -28.1933 \\ 28.1933 & -221.8223 \end{bmatrix} \tag{32}$$

To apply this gain constant in the system, the A matrix must be replaced by (A-B*K). Let all the other state matrices be the same and run the simulation again. The system becomes stable, but the response is nowhere close to being usable at this point. Figure 7 shows the output response, and the settling time is somewhere around 70 seconds.

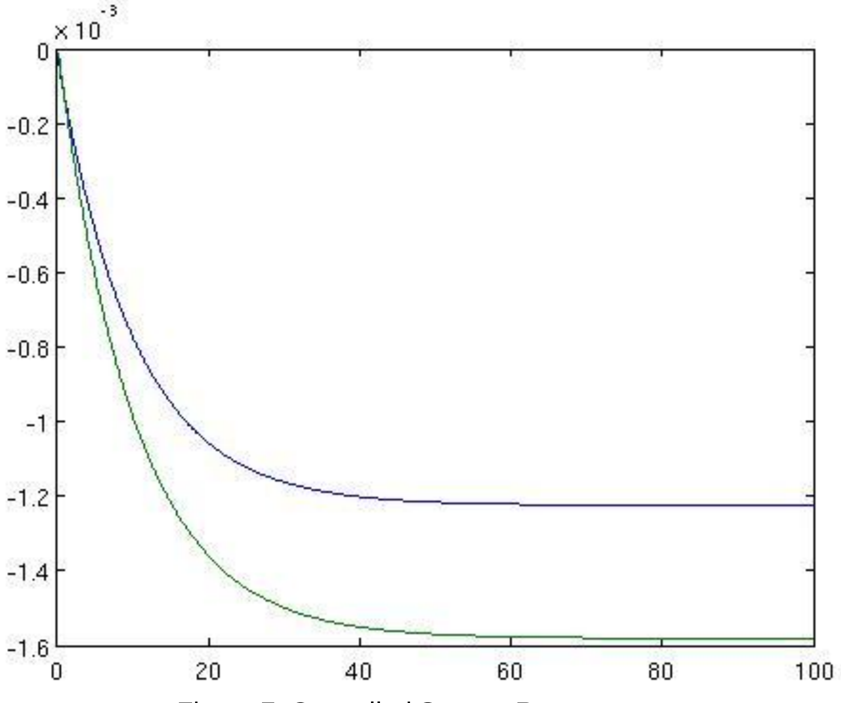


Figure 7: Controlled System Response, K_1

Since the response time of a system can be modeled using an exponential as shown in (33), increasing K and thus moving the poles farther left will force the system to respond faster.

$$y(t) = Ce^{(A-BK)t}x(0) \tag{33}$$

Therefore, increasing Q and recalculating K will increase K. So an arbitrary value of 10 was chosen to multiply Q by. Figure 8 shows the response of the system after the Q value was increased. Also, it was notice at this point that since Q is a scalar multiple of the 2x2 identity matrix, the first value of the matrix K can be used instead of the entire matrix. Thus

$$K_2 = -2.2182e3 \tag{34}$$

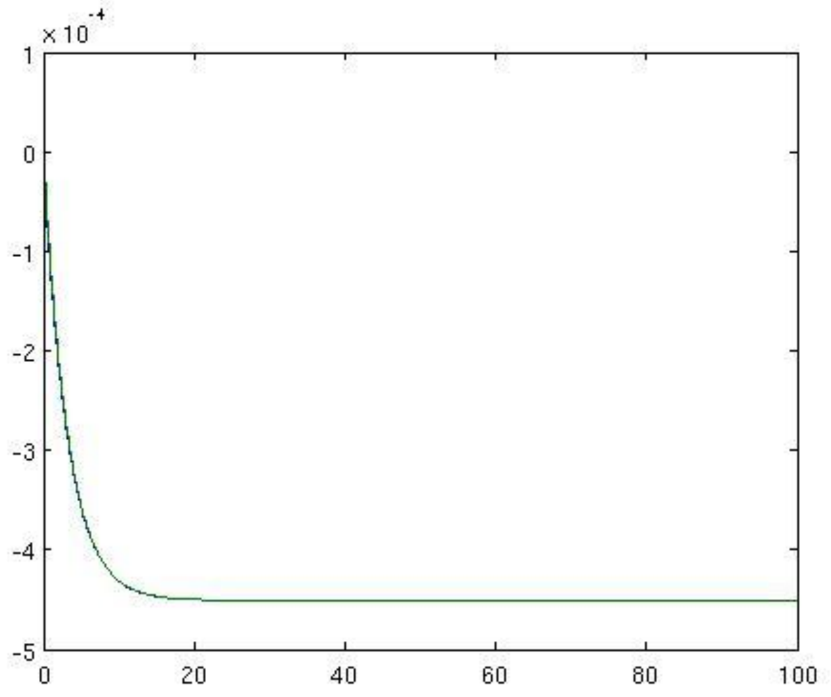
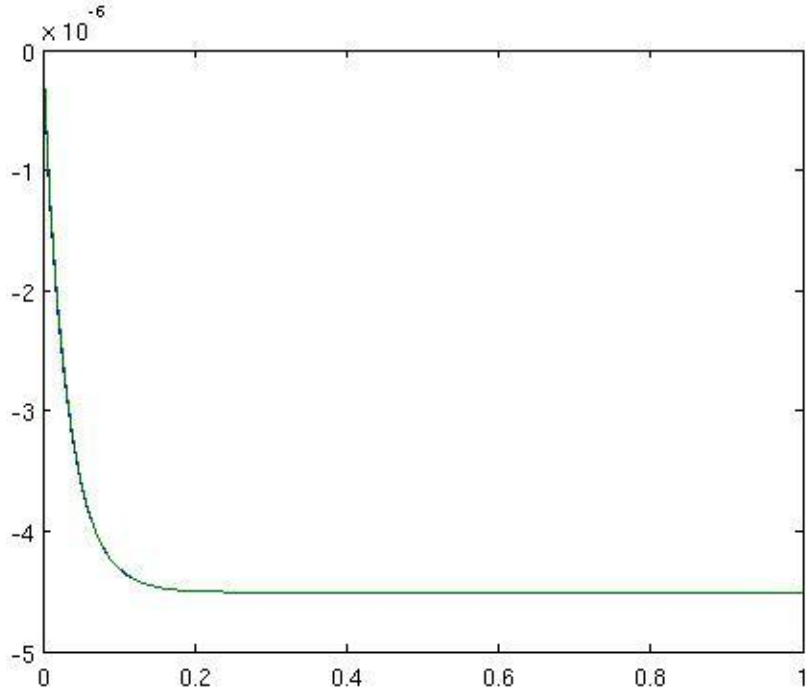


Figure 8: Controlled System Response, K_2

This still isn't the desired response time, so one more drastic measure. The next step taken was to multiply K by 100 to give a rather large increase in the exponential decay of (33). Figure 9 shows a desirable settling time of about 0.2 seconds.

$$K_3 = -2.2182e5 \tag{35}$$



The next step taken was to set a reference input value so the full state back controller will have an output that tracks the desired input. The reason that a reference input needs to be instated is that we are not comparing the output with the desired input value. We are instead measuring all state values, multiplying them by the gain factor K and then finding the difference of this value from the desired input value. There is no reason to expect K times the state values to be equal to the desired input. In order to eliminate this issue, the reference input is scaled to make it equal to the feedback steady state. Since it would be nice for the response to mimic the input, the reference input value was set to the K gain. This should bring the output value close to the desired input value. To implement this in the MATLAB state-space function, multiply the B matrix by the reference K value. In this case

$$r = K_3 \tag{36}$$

$$B * r = B * K_3 \tag{37}$$

Figure 10 shows that the response now tracks the desired input very well.

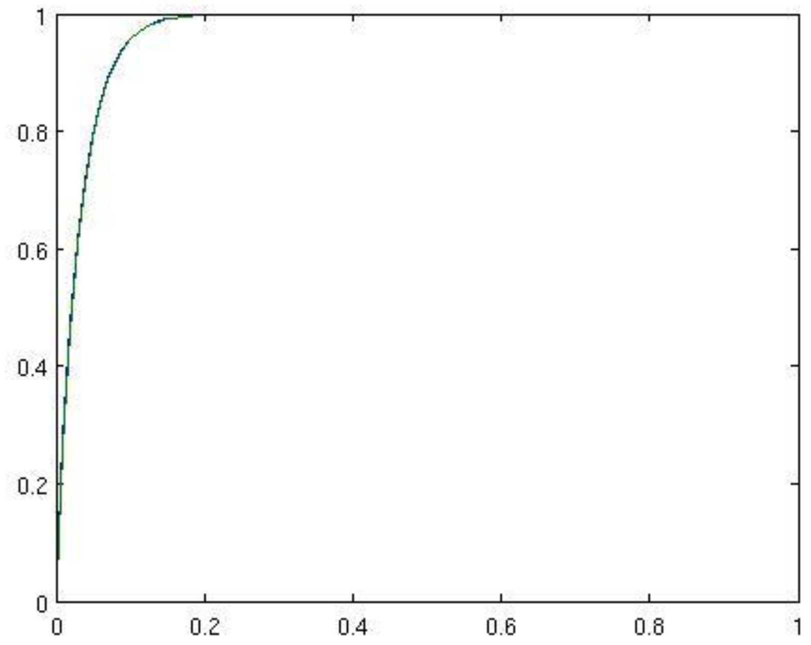


Figure 10: Controlled System Response, with reference

A. Simulink Model

This section does not add to the work already done in the MATLAB section. Simulink is a program that allows the user to create and run systems based on block diagrams. It is of the opinion of this paper that block diagrams help visualize the flow of information and significantly helps in understanding how a state variable model functions. Therefore, simulink models of the system were made to help in the understanding of this system.

First, figure 11 shows a basic block diagram model. This model was built utilizing all the state matrices for completeness. This is then run in the Simulink simulator utilizing the same conditions as the MATLAB code. The response of the uncontrolled state block diagram is shown in figure 12. It is clear that this system is also quite unstable.

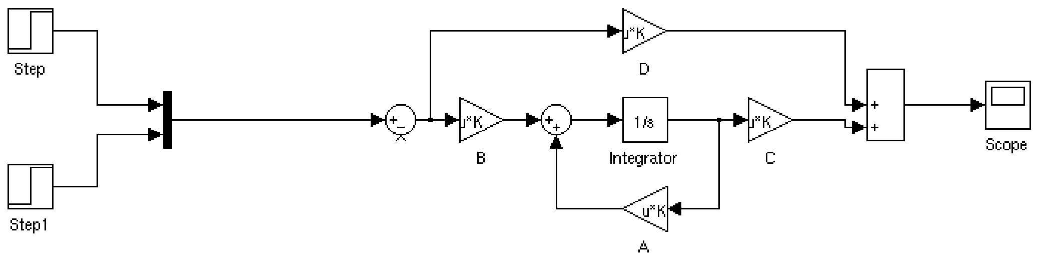


Figure 11: Uncontrolled State Block Diagram

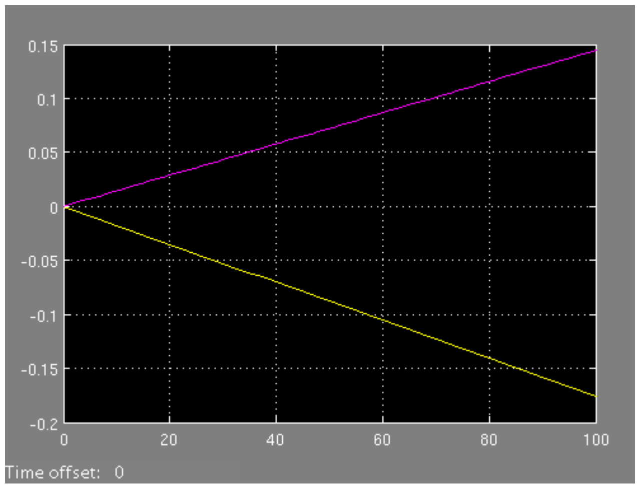


Figure 12: Uncontrolled State Block Diagram, Response

In order to simulate a state variable model in a block diagram, the knowledge of how the flow of a state system works. First there is the input value. This value gets multiplied by the B (input) matrix. The term gets added to the state variable matrix that has been multiplied by the A (system) matrix. The value then gets integrated and becomes the current state variable. Finally it is multiplied by the C (output) matrix. The D (feedforward) matrix usually is set to zero, but there are some cases where the output of the system is being modified by the previous input of the system. This is where the feedforward name comes from. The input is “feed” forward to the output of the system.

Now that there is an understanding of how the state variable system works, the gain constant is added to stabilize the system. Figure 13 shows the block diagram, while figure 14 shows the response of the system.

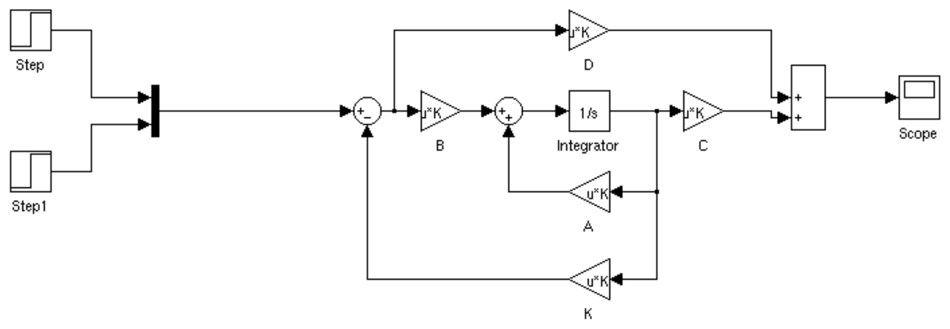


Figure 13: Controlled State Block Diagram no reference

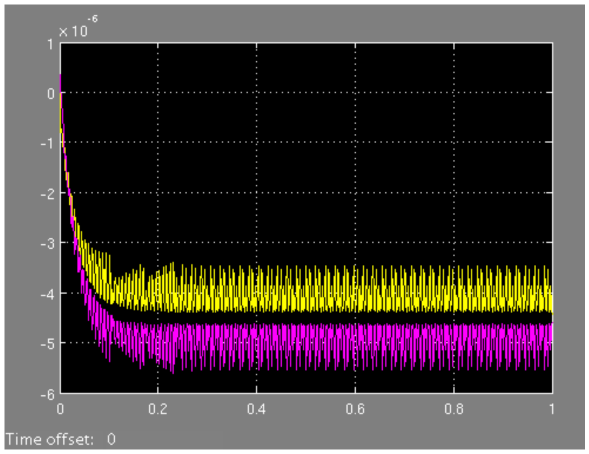


Figure 14: Controlled State Block Diagram no reference, Response

The gain constant, K, was placed leading out from the current state variable line and it is “feeding” it back to start of the state block diagram. This feedback loop is then subtracted from the desired input.

The final block to add in our state block diagram is the reference input. This block will go before the subtraction block of the gain constant, K. This reference, just like in the MATLAB code, will force the output to track the desired input. Figure 15 shows the block diagram of the state system with the reference gain added and figure 16 shows the response of the system.

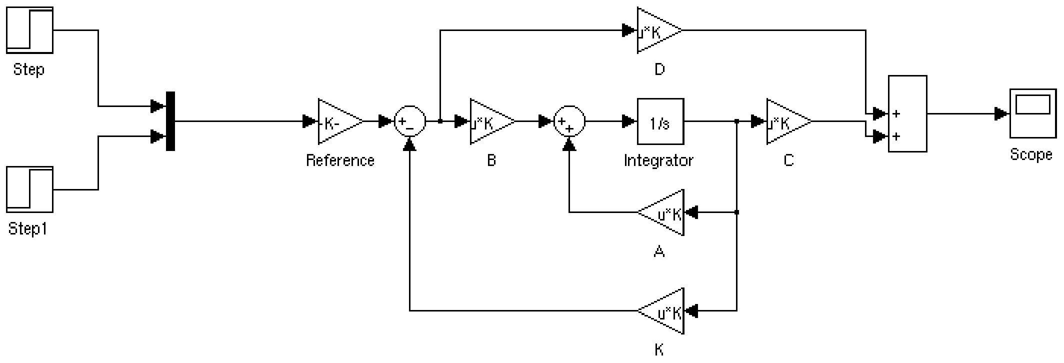


Figure 15: Controlled State Block Diagram with reference

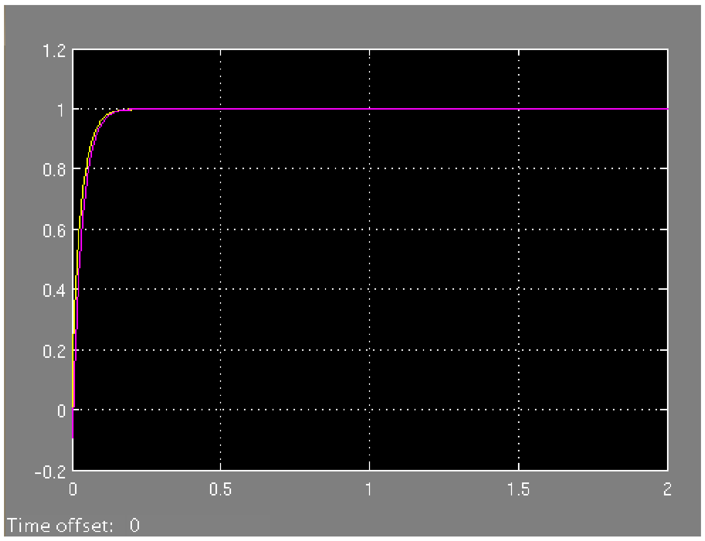


Figure 16: Controlled State Block Diagram with reference, Response

VI. **Conclusion**

There is a growing demand for autonomous mobile grasping robots in industrial settings as well as for public use; from robots that can be utilized in multiple parts of an industrial work place to adaptable interaction robots that will be able to assist humans in their everyday lives. In order to accomplish controlling such robots, a clear understanding in multidisciplinary areas are required including: kinematic modeling for a robot, computer vision and how it can relate to the physical world, mathematic manipulation, and control theory. In this paper, a breakdown of each area was presented as well as how they relate to one another. This was then followed up by the formation of a control law and simulated implementation. Once the concept and the understanding of how state feedback control functions, the development of a control is quite easy. A basic idea of where to place the poles of the system was used. Then some trial and error lead to the developed controller above.

I have learned quite a lot about control systems, control theory, and applications of control theory during this course of this project. The main focus of this project was to develop a full state feedback controller for a system developed in [1]. I believe I have achieved creating a full state feedback controller which will stabilize and control the mobile robot system discussed in the beginning of this paper. Though I wished to have accomplished more with this project, time constraints put on by the heavy work load this semester lead me to only complete some of the goals I set forth in the beginning of the semester. Throughout the entire process of developing the full state feedback controller, I learned robotic systems, kinematics, Linear Algebra, space transformations, basics of computer vision, some advanced subjects in State Space Modeling, and a brief glimpse of what modern control theory can offer. The information that I obtained from the process of this self guided project will help strengthen my ability to work on larger projects in the future.

Future work for this project will be to test and modify this controller on a Pioneer P3-DX robot. Once the testing and reworking phase is completed, a visual reinforcement learning algorithm will be developed to determine the optimal grasping location to place the robot's endeffector in order to pick up various objects. This work will be continued in Dr. Wang's research group.

REFERENCES

[1] Y. Wang, H Lang, C. W. de Silva, ”A hybrid Visual Servo Controller for Robust Grasping by Wheeled Mobile Robots,” *IEEE/ASME Trans. on Mechatronics*. 2009.

[2] S. Hutchinson, G. D. Hager, and P. I. Corke, “A tutorial on visual servo control,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[3] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. NewYork: Wiley, 2006.

[4] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Cambridge, MA: MIT Press, 2004.

[5] Picture. http://en.wikipedia.org/wiki/PID_controller

[6] http://www.engin.umich.edu/group/ctm/examples/pend/invSS.html

[7] Wolfram Mathematica. “Linear Quadratic Regulator.” http://reference.wolfram.com/legacy/applications/control/OptimalControlSystems Optima/10.1.html

APPENDIX A

```
%------------------ Figure 6 ----------------------------------------
% call getMDL and Set State Matrices
[M DL] = getMDL();
B = M*DL;
A = zeros(2);
C = eye(2);
D = zeros(2);
sys_nc = ss(A,B,C,D);

% Test the uncontrolled system to show non-stability
T = 0:0.1:100;
U = [ones(1,1001);ones(1,1001)];
[Y,T,X] = lsim(sys_nc,U,T);
plot(T,Y);
%------------------ Figure 6 ----------------------------------------

pause;

%------------------ Figure 7 ----------------------------------------
% Set Q and R for Linear Quadradic Regulator funcion
Q = [500000,0;0,500000];
R = eye(2);

% Find K gain value based on LQR, e = pole values
[K,S,e] = lqr(A,B,Q,R);

% Now to make new State Space model with Gain value added
sys_cl = ss((A-B*K),B,C,D);
Ac = (A-B*K); Bc = B; Cc = C; Dc = D;

% Set time a variable T to range from 0 to 100s at 0.1 steps
% Set a unit input of 1 by way of a variable U
T = 0:0.1:100;
U = [ones(1,1001);ones(1,1001)];
[Y,T,X] = lsim(sys_cl,U,T);
plot(T,Y);
% the plot shows the system is being controlled, but not very quickly
% Lets place the poles farther to the left
%------------------ Figure 7 ----------------------------------------

pause;

%------------------ Figure 8 ----------------------------------------
% Try multiplying Q by 10
```

```
Q = Q*10;
[K,S,e] = lqr(A,B,Q,R);

sys_cl = ss((A-B*K(1)),B,C,D);
Ac = (A-B*K(1)); Bc = B; Cc = C; Dc = D;

T = 0:0.1:100;
U = [ones(1,1001);ones(1,1001)];
[Y,T,X] = lsim(sys_cl,U,T);
plot(T,Y);
% Better time, but not quite quick enough
% Now try moving the poles even further to the left
%------------------ Figure 8 -----------------------------------------

pause;

%------------------ Figure 9 -----------------------------------------
% Multiply K by 100
K1 = K*100;

sys_cl = ss((A-B*K1(1)),B,C,D);
Ac = (A-B*K1(1)); Bc = B; Cc = C; Dc = D;

T = 0:0.01:1;
U = [ones(1,101);ones(1,101)];
[Y,T,X] = lsim(sys_cl,U,T);
plot(T,Y);
%------------------ Figure 9 -----------------------------------------

pause;

%------------------ Figure 10 -----------------------------------------
% Now to add a reference input to get a better output
sys_cl = ss((A-B*K1(1)),(B*K1(1)),C,D);
Ac = (A-B*K1(1)); Bc = (B*K1(1)); Cc = C; Dc = D;

T = 0:0.01:1;
U = [ones(1,101);ones(1,101)];
[Y,T,X] = lsim(sys_cl,U,T);
plot(T,Y);
% So, the output is where we want it
%------------------ Figure 10 -----------------------------------------
```

```
function [ M, DL ] = getMDL()
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
r = 600;
c = 300;
rd = 334;
cd = 400;
theta = 0;
lam = 0.01;
zc = 1;
sx = 0.000005;
sy = 0.000005;
dx = 0;
dy = 0.1;
dz = 0.3;
oc = 254;
or = 334;
D = 0.2;
l = 0.4;

u = -sx*(r-or);
v = -sy*(c-oc);

M = [u/zc, ((u*v*sin(theta))-((lam^(2)+u^(2))*cos(theta)))/(lam) -
((lam*dz*cos(theta))/(zc)) - (u*dx)/zc; v/zc, (((lam^(2)+v^(2)) -
u*v*cos(theta))/(lam)) - ((lam*dz*sin(theta))/(zc)) - (v*dx)/zc];

DL = [(D/2), (D/2); (D/l), -(D/l)];


End

ans =

   -0.0013   -0.0132
   -0.0002    0.0100
```