Ball and Beam Balance Final Report

Tom Trapp and Wil Selby
Department of Mechanical Engineering
Massachusetts Institute of Technology
2.151 Advanced System Dynamics & Control
Fall 2009

Abstract

The primary objective of this project was to create a control system that could effectively balance a ball on a metal beam using a motor input to control the angle of the beam. This project covered the full scope of design including creating a valid model of the system, identifying numerical values for the model parameters, designing an effective control system, assembling the system hardware, and implementing the control system using a microcontroller to interface with the motor. The ball and beam balance problem is a classic open loop unstable system. For a constant input there is a non-constant output. In this system, a constant beam angle causes the ball to accelerate due to the force of gravity and the ball's position increases non-linearly. Investigating this system resulted in numerous insights of control design that can be applied to multiple applications. Utilizing full state feedback, the controller was able to repeatedly and effectively balance the ball at the given desired reference point with reasonable transient performance. In the future, the addition of a Kalman filter for state estimation and a LQR based controller could make the system even more effective.

I. Introduction

The ball and beam balance system is one of the most popular laboratory controls design experiments because it is very simple to understand and can be used to study the implementation of classical and modern control techniques. The system is fairly straightforward. A motor is attached to the beam in a way that the shaft of the motor can control the angle of the beam. This is usually done with the use of a lever arm attached to the end of the beam or with the motor shaft connected directly to the center of the beam. For ease of assembly and troubleshooting, it was decided to directly couple the motor shaft at the center of the beam. Once assembled, the motor and beam angle are used to move a ball placed on the beam to a desired position. Feedback usually comes in the form of an encoder to measure the motor position and a resistive wire or acoustic sensor to detect the ball's position on the beam.

While the system doesn't require any complex assembly, the tools used to evaluate and control the system can be advanced and applied to other more complex systems. The ball and beam balance is a safe way to investigate control design for unstable systems. The ball and beam system is open loop unstable since for a given constant angle of tilt on the beam, the ball's position changes without limit. Therefore, some form of feedback loop must be used to control the ball's position. While the ball and beam system is not a model of a real system, its dynamics are similar to some of the most challenging areas of modern control. The dynamics of the ball and beam system are specifically similar to aerospace systems such as the control of vertical thrust in rockets or vertical take-off aircraft. The angle of the thrusters must be constantly controlled in order to keep the rocket or aircraft moving in a constant direction. Unstable systems are also seen in the chemical process industries in exothermic reactions. As the chemical reaction

occurs, heat is produced. However, the heat also increases the rate of the reaction so some form of control is needed to stabilize the reaction process. Many other areas of industry have similar open loop unstable control problems, which makes the study of the ball and beam balance relevant.

II. System Model

A simple depiction of the ball and beam balance system can be seen in Figure 1.

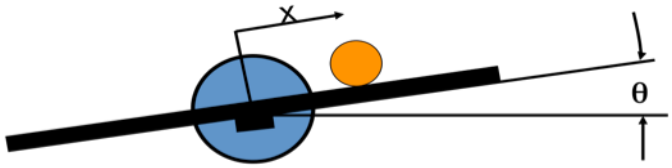


Figure 1: Ball and Beam System Model

The ball's position is measured relative to the center of the beam with positive displacement to the right of center. Therefore, when control was implemented, the final desired ball position was at x=0. Also, beam angle (theta) was defined with positive theta being produced by a counter clockwise displacement of the beam. Since the motor is directly coupled to the center of the beam, the motor position angle and the beam position angle are identical. To make the system easier to troubleshoot, a ball was replaced with a spool that matched the width of the beam. This forced the spool to only move in the x direction.

For this system, the model was developed using two separate transfer functions and then combining them for a full system transfer function. First, a transfer function for the ball and beam was derived relating an input of theta to a ball position x. The complete derivation can be seen in Appendix 1 and the final transfer function is shown in Equation 1 below. The ball and beam dynamics are nonlinear in nature, however, using a small angle approximation, the dynamics can be linearized and form the equations below.

$$\frac{X(s)}{\theta(s)} = \frac{m_s g}{\left[m_s + \frac{J_s}{l^2}\right]s^2}$$

Equation 1: Ball and Beam Transfer Function

The variables are defined as:
m_s = mass of the spool
J_s = inertia of the spool
l^2 = effective rolling radius (distance from the spool center to the point of contact with the beam)
g = gravity

This transfer function contains a double integrator which places two poles at 0. This is clearly an unstable system. The transfer function is represented in state space form in Equation 2 below.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{m_s g}{\left[m_s + \dfrac{J_s}{l^2} \right]} \end{bmatrix} \theta$$

$$Y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

Equation 2: Ball and Beam State Space Model

Next, a transfer function relating a motor voltage input to motor/beam position theta was derived. The full derivation can be referenced in Appendix 2. The transfer function is reproduced below in Equation 3.

$$\frac{\theta(s)}{V(s)} = \frac{K_t}{s^2 J_{sys} R + s K_t^{\,2}}$$

Equation 3: Motor System Transfer Function

The variables are defined as:

K_t = motor torque constant

J_{sys} = moment of inertia of the ball, beam, and motor system

R = armature resistance

The transfer function was then rewritten in state space form as shown below in Equation 4.

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \dfrac{-K_t^{\,2}}{J_{sys} R} \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{K_t}{J_{sys} R} \end{bmatrix} V$$

$$Y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

Equation 4: Motor System State Space Model

These two systems were combined to compute an overall transfer function that related a motor voltage input to an output of ball position. The transfer function can be seen in Equation 5 as well as its state space representation in Equation 6.

$$\frac{X(s)}{V(s)} = \frac{m_s g K_t}{s^3 \left[m_s + \dfrac{J_s}{l^2} \right] \left[s J_{sys} R + K_t^{\,2} \right]}$$

Equation 5: Ball, Beam and Motor System Transfer Function

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{m_s g}{\left[m_s + \frac{J_s}{l^2}\right]} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \dfrac{-K_t^{\,2}}{J_{sys} R} \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dfrac{K_t}{J_{sys} R} \end{bmatrix} V$$

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

Equation 6: Ball, Beam and Motor System State Space Model

The complete system allows us to combine the dynamics into one transfer function as well as one state space system. This is a four state system with the states being ball position and velocity as well as beam angle and velocity. This model uses voltage as an input instead of current because the microcontroller outputs a voltage command to the motor. This made it straightforward to implement the controller later in the project. Using a voltage command input to the system, the ball's position could be controlled.

Before designing a controller, it was necessary to get numerical values for the parameters of the model. As mentioned previously, a spool was used instead of the traditional ball in order to ease troubleshooting of the system. For the model, it was necessary to find the inertia and mass of the spool as well as the spool's effective rolling radius. The complete derivation of these numbers can be referenced in Appendix 3. The mass of the spool was 9.4 g, the inertia was 9.06 mg-m^2, and the effective rolling radius was 1.46 cm. Gravity was taken as 9.8 m/s^2. For the motor system, the motor torque constant was listed in the motor data sheet (Appendix 4) as .0332 N-m/A and the armature resistance was listed as 25 Ohms. Lastly, the inertia for the entire system was calculated. The motor and gearbox inertias were listed on the data sheet in Appendix 4. The beam inertia as well as the tilt sensor inertias were calculated as can be seen in Appendix 3. The final system inertia was 3.7994e-07 kg-m^4. With these values calculated, the model was complete and a controller was designed. The final open loop system had the transfer function and state space values shown in Equation 7 below.

$$\frac{X(s)}{V(s)} = \frac{6207.0014}{s^3(s+116)}$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1.7758 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -116.0426 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 3.4953 \times 10^{-3} \end{bmatrix} V$$

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

Equation 7: Numerical Ball, Beam and Motor Model

After finalizing the model, the appropriate sensors and actuators were chosen in order to provide the input to the system and measure the desired states for use in the control feedback. The motor was chosen because it had ample torque to drive the system without limitation but yet did not draw more load current than the motor control board could provide. To sense the ball's position a PING acoustic range finder was used. Specific details about this device can be seen in Appendix 5. Lastly, a method to determine the beam angle was needed. Initially, the motor encoder was used to determine the motor position. In order to measure the angle exactly, every encoder count needed to be logged. Unfortunately, the microcontroller had problems reading in the encoder data accurately so the motor encoder could not provide the accuracy needed for the control loop. Instead, a tilt sensor was placed on the beam to accurately measure the angle. This sensor provided a voltage from 0 to 5V that could be used to calculate the beam angle very accurately. As seen in the data sheet in Appendix 6, the specific formula (Equation 8) was non linear. The angle was in degrees and the offset was the motor voltage when the sensor was level. This was experimentally determined to be 2.4V. This sensor supplied very accurate angle measurements and the microcontroller resolved the signal to approximately .4% (1÷256).

$$angle = \arcsin\left(\frac{V_{out} - Offset}{2}\right)$$

For design simplicity, our model assumed that both sensors had perfect dynamics and responded instantly. In practice, our final sensor suite proved to be very reliable and accurate and the control routine was limited by the microprocessor speed more than the sensor dynamics. The derivative states were found by taking the derivative of the sensor measurements of ball position and beam angle respectively. Future work on this project could include an observer or estimator system to provide more accurate values for these states.

III. Controller Design

The first step in designing an effective control system for this system was to analyze the open loop response. The complete MatLab code can be seen in Appendix 7. As described previously, the system is open loop unstable and for our model, the root locus of the system is shown below in Figure 2.

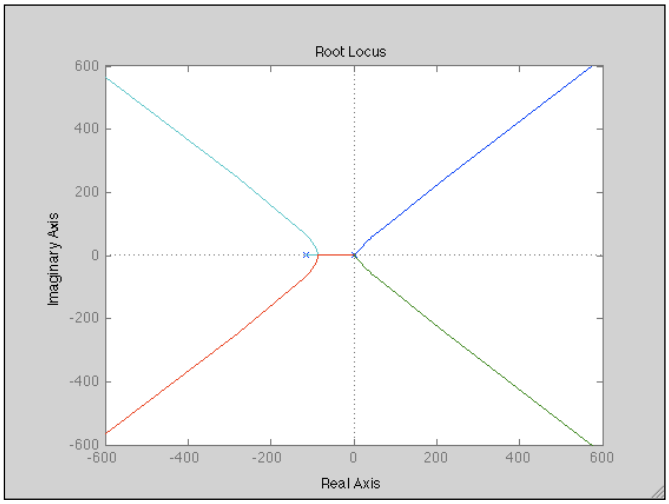


Figure 2: Ball and Beam Balance Open Loop System

This system is clearly unstable as two poles move from 0 to the right half plane as K is increased. Feedback control is necessary to place all poles in the left half plane of the root locus plot. Two separate control strategies were investigated for this project. A full state feedback controller using state space design methods was utilized for this project. For the transient performance, the system was desired to have a response of 0.8 seconds settling time, less than 4.33% overshoot, and 0% steady state error. The controllability and observability matrices were also calculated and both had full rank as well. This proves that the system is completely state controllable and state observable.

By examining the root locus, it is evident that a pure proportional controller will not be able to control the system. Instead, a full state feedback controller was utilized. This had several advantages over classical control techniques. Primarily, state space design allows for specific pole placement which makes designing a controller for specific transient response much easier. Also, with the knowledge that this controller would be implemented utilizing a microcontroller, using full state feedback allowed for easier implementation as well as tuning of the specific gains. Lastly, given the fact that our sensors could read ball position and beam angle accurately, it was thought that computing the translational and angular velocities derivatively would be reliable in the absence of an observer or estimator. Using state space methods also allowed for the investigation of optimal control methods to determine the gain matrix, however this approach was not utilized in this project.

Using the A,B,C, matrices listed in Equation 7 as well as 0 for the D matrix, the 'place' command in MatLab calculated the gains of the K matrix. With the design requirements listed above, the dominant poles were calculated to be $-5 \pm 5j$. Since the system is fourth order, two other poles had to be specified. These were chosen as -20 and -60 so that these poles would have little effect on the overall system response. With these desired poles, the K matrix was [9.6665 2.5777 0.5865 -0.0075]. These poles were checked by finding the eigenvalues of the closed loop A matrix, A-B*K. The eigenvalues matched the desired pole locations, so the gains were validated.

Next, the closed system was simulated with a step response to determine the transient response. For this system, a step response physically would be to place the ball at the center of the beam with no control being applied to the motor. The step input would be forced by changing the desired ball position from 0 to 1 unit away. For simulation purposes, a step of .05m was

applied to the system. A plot of the step response is shown below in Figure 3 as well as the block diagram used for simulation in Figure 4.

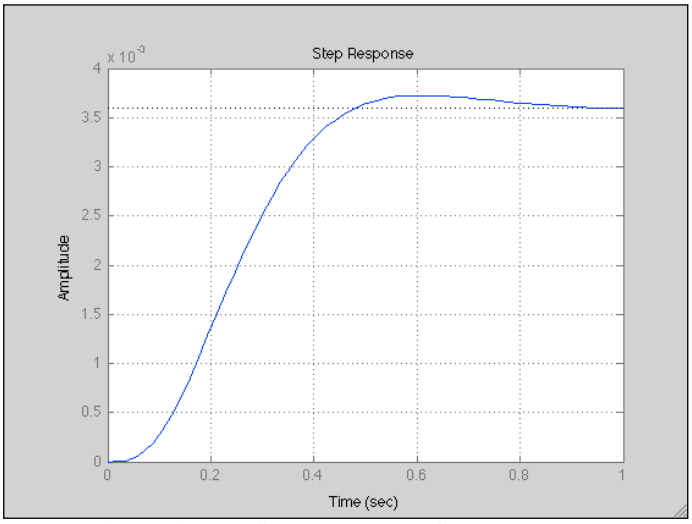


Figure 3: Full State Feedback Step Response

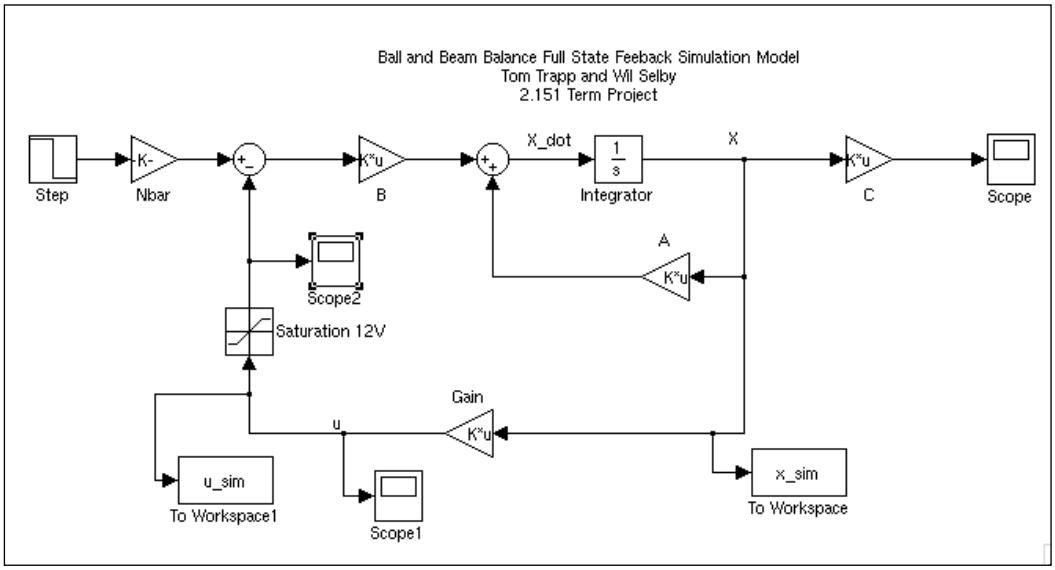


Figure 4: Ball and Beam System Simulink Diagram

As evident by the response in Figure 3, the design meets the overshoot requirement, but the settling time is about 0.1 seconds too slow and there is significant steady state error. While we did place the dominant poles to match our desired response, the other two poles placed farther down the real axis do have a slight effect on the system response. Therefore, the dominant poles were moved to $-6 \pm 6j$ to compensate for the effect of the other poles and to reach the original design specs. When this system is simulated, there is still extensive steady state error, but the overshoot is 3.73% and the settling time is 0.769 seconds, both within the original design specifications. For this set of desired closed loop poles, the K values were [13.9198 3.2479 0.6386 -0.0069]. The combination of these gain and the previous set created a window of

performance that was useful when the controller was tuned later in the project. To validate the K values, a quick LQR command was done in MatLab which produced K values very close to the ones found with the 'place' command.

Next, it was necessary to implement a reference input gain to eliminate the steady state error of the system. In classical control, the output is compared to the reference input in a summing junction to compute an error signal that is then fed into the compensator. However, as evident by the block diagram in Figure 4, full state feedback multiplies each of the states by a gain value which can't be compared to the reference input. To solve this, a reference input gain called the feedforward gain (Nbar) is calculated to scale the reference input and make it equal to K*x_steadysate. For this model of the system, the value of the feedforward gain is always the value of the position state gain. With the feedforward gain placed in the model, as seen in the Simulink diagram in Figure 4, the step response (Figure 5) has a steady state value of .05 and the steady state error has been eliminated.

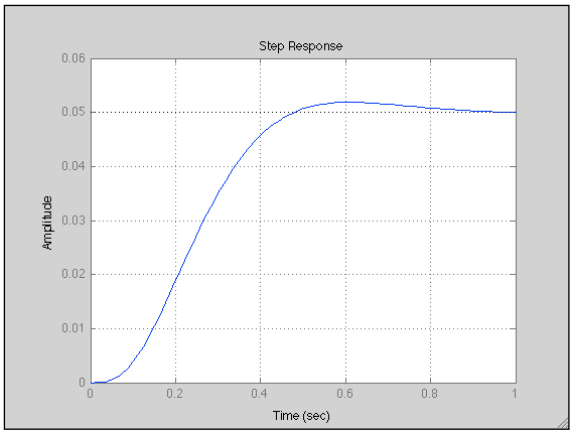


Figure 5: Step Response With Feedforward Gain

Once satisfied with the step response transient response, the control output was plotted to make sure that the motor would not be saturated. The maximum motor voltage is 12V and as evident by the plot below, the maximum voltage required by the control system is around 1V, well under the motor limits.

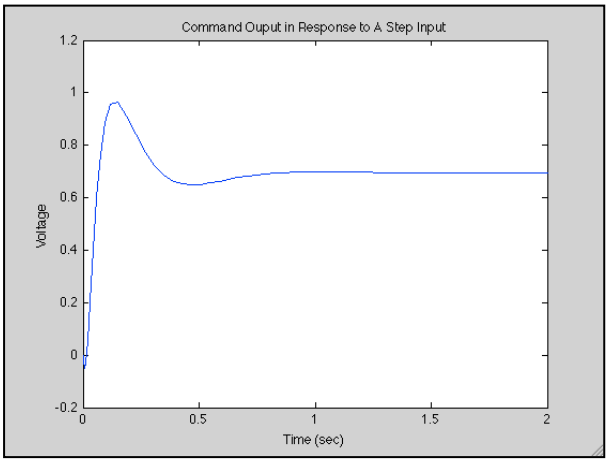


Figure 6: Control Output to Step Response

IV. Results from Simulations and Experiments

With the controller effectively designed in simulation, it was necessary to implement the gain values found into the physical system. To do this, the Arduino microcontroller board was used with a motor shield. Pictures of these devices can be seen in Appendix 9. Using a microcontroller allowed for easy tuning of the gain values and offered more flexibility compared to an analog controller. The Arduino was also interfaced with the tilt sensor and the PING rangefinder to read in the state values of ball and beam position. The final code used to run the controller can be seen in Appendix 8 and a simplified schematic shown below in Figure 7.

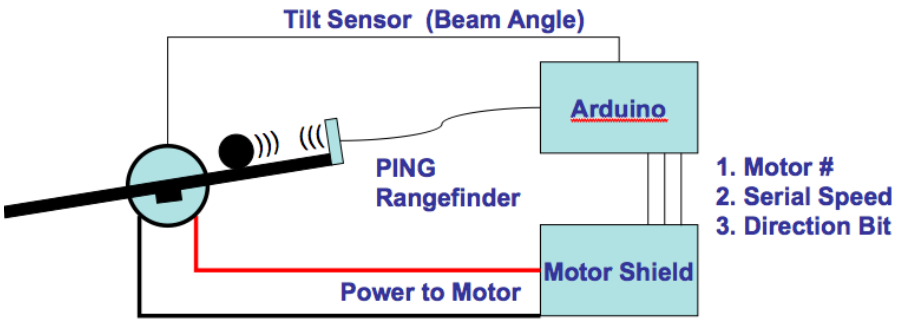


Figure 7: Hardware Schematic

There were several issues that had to be overcome when implementing the controller. Primarily, the motor encoder would not give reliable motor position measurements. The encoder was a quadrature encoder which outputs two out-of-phase square waves. By reading the values of the two waves when one wave went from high to low, it was possible to tell the direction that the motor was spinning and how many revolutions the motor had completed. In order to be accurate, every change from high to low had to be counted. Traditionally, this is done in software using an interrupt which monitors the signal and runs user specified code when a change is detected in the encoder output. When implemented, it appeared that the interrupt was being triggered at various times, seriously degrading the accuracy of the encoder measurement. This also proved to be more inconsistent as the motor speed was varied. Since the control output would command the motor to operate at various speeds, the encoder data could not be reliable. After further investigation, it was believed that it was necessary to debounce the signal or that the encoder itself was malfunctioning. The concept behind debouncing is simple. When the state of each signal changes from high to low, it does not do so instantaneously. Instead, there is some time constant where the voltage ramps from low to high voltage or vice versa. Digitally, a value below a certain voltage is interpreted as a 0 and a voltage above a certain threshold is given a value of 1. However, when the voltage is in the area between those thresholds, the computer can't decide if the value is a 1 or 0. The solution is to wait some small amount of time from the instant the change is detected to confirm that, for example, the change from low to high did result in a high value and wasn't just noise in the signal. This strategy was attempted, but since every change had to be detected, a delay in the reading of the signal caused changes to be missed and accuracy sacrificed. The solution was to use a tilt sensor to measure the beam angle which performed very well.

To measure the ball's position, the PING rangefinder was used as mentioned previously. Several limitations of this sensor had to be overcome as well. Initially, the plan was to use the full length of the beam, about a meter. Unfortunately, the geometry of a spherical object resulted in a range of about half that distance. Attempts to use a golf ball or other objects did not improve the sensor range dramatically. Therefore, the rangefinder was moved to a position halfway between the beam end and beam center. The PING could identify objects this distance with no obstructions. However, when the ball passed the center of the beam, the rangefinder would lose the ball. Instead, the PING was focusing on the coupling that connected the motor shaft to the beam. The PING was then moved back to the end of the beam and the ball was prevented from moving past the center of the beam. This was the final configuration of our set-up and once this was implemented, the PING had no trouble identifying the ball on the beam.

Once the hardware issues were overcome, code had to be written in the Arduino syntax to read in the sensor signals, compute the control output, and send it to the motor. Interpreting the Arduino syntax and unique functions created a small learning curve, but implementing the code in a modular fashion helped the debugging process. The tilt sensor was read in as an analog voltage which the Arduino converted to a number from 1 to 256. This then had to be converted back into a voltage and converted to an angle using the previously mentioned formula. The PING rangefinder sent back a digital pulse whose width corresponded to the distance to the target. That time duration along with the speed of sound was used to compute the range in centimeters. Specific details can be seen in the datasheet in Appendix 5.

The next issue that had to be addressed for the use of full state feedback was state estimation. Instead of designing an observer, it was decided to compute the time derivative of the positions to find the state velocities. This was much easier to implement while learning how to effectively code in the Arduino environment, however, accuracy was probably sacrificed. As evident in the code, the position outputs from the sensors were filtered and then divided by the control sample time to determine velocities.

Once all states were measured, the control calculation could be computed. One advantage of full state feedback is the ease of implementation in software. For classical control, the compensator must be converted to the z domain and then a difference equation would need to be computed to compute the motor input. Any change in the compensator results in rederiving the difference equation. Full state feedback allows simple edits since only the gain values themselves are changed. Each gain is multiplied by its specific state and the sum is the motor control value. In the Arduino environment, the motor is controlled by a pulse width modulated signal. Thus, the control output had to be converted to a PWM signal by the conversion 255/12 (counts/volts) which gives the PWM controller a duty cycle percentage.

Lastly, timing issues had to be sorted out to get the code to run effectively. Initially, the code was designed to run at 100Hz. Using the onboard timer, the time to run the code was calculated and then a delay was used to hold the routine until the desired sample time was reached. Initially, serial communications were used to print the values of the states and the control commands to a console. This caused the control routine to run much slower, about 10 Hz. The control routine could not respond fast enough to the system and the system was unstable. When the output commands were deleted, the control routine ran at the desired frequency and the control routine was able to stabilize the system.

As mentioned previously, the additional closed loop poles resulted in less than desired transient performance. The dominant gains were modified to create a K matrix that performed to specifications in simulation. Using the two pairs of K values as guidelines, the controller was

tuned until a desired stable response was achieved. Using the initial state of K values resulted in an unstable closed loop system. The main reason for this inaccuracy was probably due to modeling errors. While our model included numerical values, the values from the data sheets were not experimentally measured. Probably the largest error of the model was due to the backlash in the motor gear head. However, slightly tuning the gains made a responsive closed loop system. In the end, the final K values were [10.7 3.5 5 -.1]. The only major modification was the value of the beam angle gain. Again, backlash in the gears likely caused some errors. These were reflected as a discrepancy between the simulated K value for beam angle and the final effective value.

Observing the system as it responded to various disturbances or step responses, it was clear that while the system was very sensitive to slight changes in the K values, the system performed well once tuned. While the initial design specifications were overly optimistic for a system with no observer and only basic pole placement control techniques, the system performed well. Figure 8 shows the results of an experimental step response.

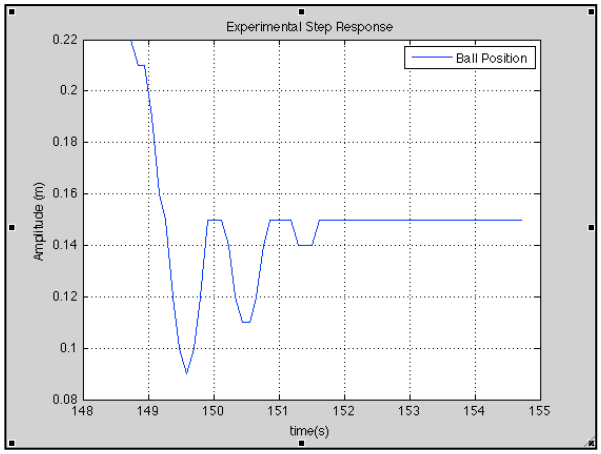


Figure 8: Experimental Step Response

The desired reference position was 18 cm away from the ping sensor. Starting the spool at .22 cm resulted in the same step input as the .05 cm step applied in simulation. In this specific example, the transient performance had 2.9 seconds settling time, 40% overshoot and -16% steady state error. Clearly this does not meet our design requirements, but the system does balance the ball around the reference point and forms a stable closed loop system. Specifically, the steady state error is most likely due to unmodeled friction in the system. There is friction between the spool and the rail which was not modeled in the system and prevents the ball from moving at very small beam angles. Also, there is internal friction in the motor and gear head which was not modeled. The motor requires a certain minimum voltage in order to turn the shaft. To fix these errors, the model could be improved. Additionally, an integral term that continually sums the ball position and beam angle error could be used to eliminate some steady state error. Our model also has been linearized around small angles of theta. For some observed responses, the beam angle reached 30 degrees which could invalidate the small angle approximation made in the model and make the control scheme less effective. To view the relative robustness of the controller, please view the attached videos which show a step response (BBB_step.mov) as well as disturbance rejection performance of the control system (BBB_distrurbance.mov). During all of our experiments with the system, the motor voltage never exceeded 8V. This means that our

controller was never saturating the motor and that it would be possible to increase the gains for increased performance in the future.

V. Conclusions and Recommendations

The ball and beam balance system is a very interesting one to study. The indirect control of ball position through control of the beam angle is fascinating to implement. Although constructing and troubleshooting the prototype was very time-consuming and we were unable due to time constraints to explore more advanced control algorithms, the project gave us great experience in hardware implementation of digital control. There is a very clear distinction from a simulation being successful and the physical system being successful. It was very beneficial to go through the entire design process from modeling to simulation and implementation. This gave us a greater understanding of what was happening physically in the system and aided the debugging process. Also, the more advanced control concepts are much easier to envision after successfully implementing full-state space control on this system. Adding gains calculated through LQR or other optimal control methods would be straightforward. Implementing an observer using the Arduino would require more software engineering research but is feasible. The main problem that lingers is the inability to use the installed motor encoder. We suspect that the problem is not with the encoder but the incorrect use of the microcontroller interrupts or possibly an issue of signal conditioning. Due to the requirement to detect every signal change, it was difficult to diagnose the specific issue with the encoder. For future study, this should be resolved. Also, although the gains we used for the final trial worked very well, the system does not seem to have the robustness or the repeatability expected from full-state space control. Perhaps there are significant unmodeled system or sensor dynamics that need to be explored. Overall, it was a very rewarding project and definitely adds intuition to the concepts learned in the course and launches us into more advanced areas of control systems design.

References

"Control Tutorials for MatLab and Simulink." University of Michigan Engineering. Nov. 2009. < http://www.engin.umich.edu/class/ctms/>.

Evanko,David, Arend Dorsett and Chiu Choi, Ph.D., P.E., "A Ball-on-Beam System with an Embedded Controller."American Society for Engineering Education. University of North Florida, Department of Electrical Engineering. Mar. 2008 <http://www.loyola.edu/betasites/cas/midatlanticasee/publications/march2008/documents/ASEE12008_0012_paper.pdf>.

Lieberman, Jeff. "A Robotic Ball Balancing Beam." 10 Feb. 2004. < bea.st/sight/rbbb/rbbb.pdf>,

Rosales, Evencio A. "A Ball-on-Beam Project Kit." Undergraduate Thesis, Massachusetts Institute of Technology, June 2004.

List of Project Tasks

System modeling and simulation – Wil 80%, Tom 20%
Prototype construction and troubleshooting – Tom 80% Wil 20%
Controller implementation and troubleshooting – Wil 50% Tom 50%
Report and presentation Wil 50% Tom 50%

Appendicies

1. Ball and Beam Transfer Function and State Space Derivation

$$\sum F_s = m_s g \sin(\theta) - F_r = m_s \ddot{x} \quad \text{where} \quad x = al$$

$$\sum \tau_s = F_r l = J_s \ddot{a}$$

$$m_s g \sin(\theta) - \frac{J_s \ddot{a}}{l} = m_s \ddot{x}$$

$$m_s g \sin(\theta) - \frac{J_s}{l} \cdot \frac{\ddot{x}}{l} = m_s \ddot{x}$$

$$m_s g \sin(\theta) = \ddot{x}\left[m_s + \frac{J_s}{l^2}\right]$$

Using small angle approximation where $\sin g(\theta) = g\theta$ leads to the equation $\frac{m_s g \theta}{\left[m_s + \frac{J_s}{l^2}\right]} = \ddot{x}$

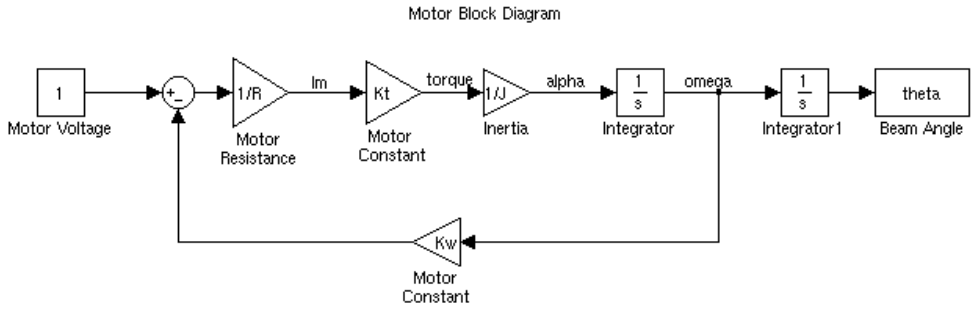
Then the transfer function relating ball position $X(s)$ to an input beam angle $\theta(s)$ is

$$\frac{X(s)}{\theta(s)} = \frac{m_s g}{\left[m_s + \frac{J_s}{l^2}\right] s^2}$$

Represented in state space as $\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{m_s g}{\left[m_s + \frac{J_s}{l^2}\right]} \end{bmatrix} \theta$

Variables:
F_s= Force on the spool
m_s= spool mass
g = gravity
F_r = rolling force constraint on the spool
x = spool position on the beam
τ_s = torque on the spool
l = distance from ball center axis to rail
a = angular displacement of spool
J_s = moment of inertia of the spool

2. Motor System Transfer Function and State Space Derivation



$$I_m = \frac{V_m - V_{emf}}{R} \quad \text{and} \quad V_{emf} = K_w \quad \text{so} \quad I_m = \frac{V_m - K_w}{R}$$

Using the block diagram:

$$I_m = \frac{V_m - V_{emf}}{R} = \frac{V_m - \frac{K_t I_m K_w}{J}}{R} = \frac{V_m}{R} - \frac{K^2 I_m}{JRs} \text{ when } K_w = K_t$$

$$sI_m R = sV_m - \frac{K^2 I_m}{J}$$

$$I_m(sR + \frac{K^2}{J}) = sV_m$$

$$\frac{I_m(s)}{V_m(s)} = \frac{s}{sR + \frac{K^2}{J}} \text{ and } \frac{\theta(s)}{I_m(s)} = \frac{K}{Js^2} \text{ then } \frac{\theta(s)}{V_m(s)} = \frac{K}{s^2 JR + K^2} \text{ and in state space form as}$$

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \frac{-K_t^{\,2}}{J_{sys}R} \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{K_t}{J_{sys}R} \end{bmatrix} V$$

$$Y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

Variables:

K_t / K_w = motor constants

J_{sys} = moment of inertia of the ball, beam, and motor system

R = armature resistance

I_m = motor current

V_m = motor voltage

V_{emf} = back emf voltage

3. Model Numerical Values

Specific values and the implementation of these formulas can be found in the MatLab simulation code below in Appendix 7.

The spool was divided into 2 pieces, the web and the center cylinder. Using the following equations, the numerical values for spool mass and intertia were calculated. R_{rot} is the distance from the center of the spool to the point of rotation of the spool when it is on the beam. This is used for the parallel axis theorem. The density of the plastic was 1.2 g/mL (PVC plastic)

$$Volume = \frac{1}{4}\pi\left(d_{out}{}^2 - d_{in}{}^2\right)\cdot thickness$$

$$Mass = \rho \cdot Volume$$

$$J_{disk} = \frac{\frac{1}{2}\pi\rho \cdot thickness\left(d_{out}{}^2 - d_{in}{}^2\right)}{2^4}$$

$$J_{spool} = J_{web} + J_{hub} + Mass_{spool}\cdot R_{rot}{}^2$$

The beam was treated as 3 individual solid cuboids to find the moment of inertia

$J = \frac{1}{12}m\left(l^2 + w^2\right)$ and $J_{beam} = 2J_{side} + J_{bottom}$

The inertia of the tilt sensor was also calculated:

$$J_{spool} = \frac{1}{2}m\left(l^2 + w^2\right) + md^2$$

where l is the length, w is the width, m is the mass of the tilt sensor calculated from density and volume as shown for the spool inertia equation above.

Finally, the system inertia was calculated

$$J_{sys} = \frac{\left(J_{beam} + J_{ts}\right)}{gear^2} + J_g + J_m$$

where $gear$ is the motor gear ratio and J_g and J_m are the motor and gear inertias listed in the motor data sheet.

4. Motor Datasheet

GLOBE MOTORS Performance Data

Globe Motor p/n 415A832

Date of Printing: 11-20-2009
Time of Printing: 09:37:07

APPLICATION DATA FILE:
MOTOR NUMBER: Model Motor

COMPUTED PERFORMANCE DATA

No Load Speed (rpm)	14.32
Stall Torque (oz-in)	220.13
No Load Current (amps)	0.11
Stall Current (amps)	0.48
RMS Torque (oz-in)	0.00
Max Efficiency (%)	18.71
Max Power Out (watts)	0.58
Max Temp (deg C)	86.40
Rated Torque (oz-in)	0.00

PERFORMANCE AT CURSOR

Speed (rpm)	12.89
Torque (oz-in)	22.01
Current (amps)	0.14
Temp Rise (deg C)	21.16
Power Output (watts)	0.21
Efficiency (%)	12.16

MODEL MOTOR CONSTANTS

Motor Voltage (volts DC)	12
Torque Constant (oz-in/amp)	4.7
Winding Resistance (ohms)	25
Temperature Rise (deg C/watt)	15
Typical No Load Torque (oz-in)	.5
Motor Inertia (oz-in-s^2)	

MODEL MOTOR GEAR BOX DATA

Gear Box Reduction Ratio (x:1)	187.68
Torque Multiplier	125.36
Gear Box Inertia (oz-in-s^2)	-

MODEL MOTOR DERIVED DATA

Motor Slope (rpm/oz-in)	0.07
Voltage Constant (Volts/Krpm)	3.48
Gear Box Efficiency (%)	66.79

5. Ping Rangefinder Datasheet

599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com
Educational: www.stampsinclass.com

PING)))™ Ultrasonic Distance Sensor (#28015)

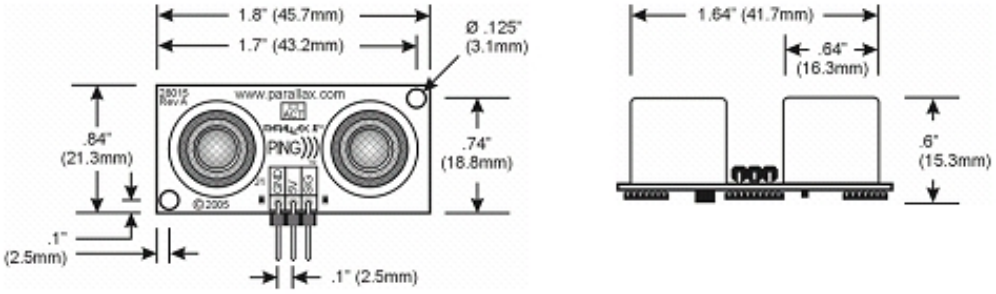
The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to BASIC Stamp ® or Javelin Stamp microcontrollers, requiring only one I/O pin.

The PING))) sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width the distance to target can easily be calculated.

Features

- Supply Voltage - 5 VDC
- Supply Current - 30 mA typ; 35 mA max
- Range - 2 cm to 3 m (0.8 in to 3.3 yrds)
- Input Trigger - positive TTL pulse, 2 uS min, 5 s typ
- Echo Pulse - positive TTL pulse, 115 uS to 18.5 ms
- Echo Hold-off - 750 s from fall of Trigger pulse
- Burst Frequency - 40 kHz for 200 s
- Burst Indicator LED shows sensor activity
- Delay before next measurement - 200 s
- Size - 22 mm H x 46 mm W x 16 mm D (0.84 in x 1.8 in x 0.6 in)

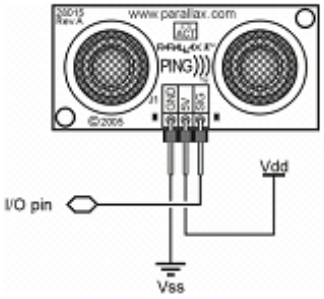
Dimensions



Pin Definitions

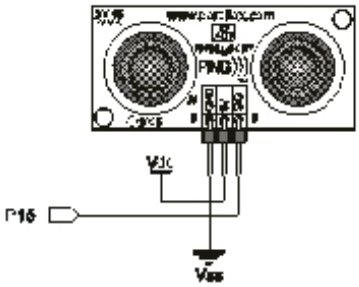
GND	Ground (Vss)
5 V	5 VDC (Vdd)
SIG	Signal (I/O pin)

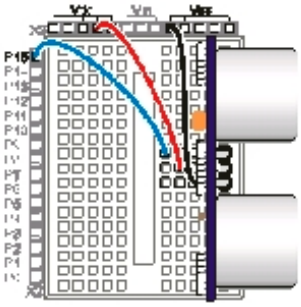
The PING))) sensor has a male 3-pin header used to supply power (5 VDC), ground, and signal. The header allows the sensor to be plugged into a solderless breadboard, or to be located remotely through the use of a standard servo extender cable (Parallax part #805-00002). Standard connections are show in the diagram to the right.



Quick-Start Circuit

This circuit allows you to quickly connect your PING))) sensor to a BASIC Stamp® 2 via the Board of Education ® breadboard area. The PING))) module's GND pin connects to Vss, the 5 V pin connects to Vdd, and the SIG pin connects to I/O pin P15. This circuit will work with the example program Ping_Demo.BS2 listed on page 7.

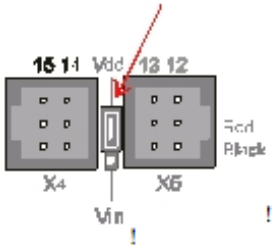




Servo Cable and Port Cautions

If you want to connect your PING))) sensor to a Board of Education using a servo extension cable, follow these steps:

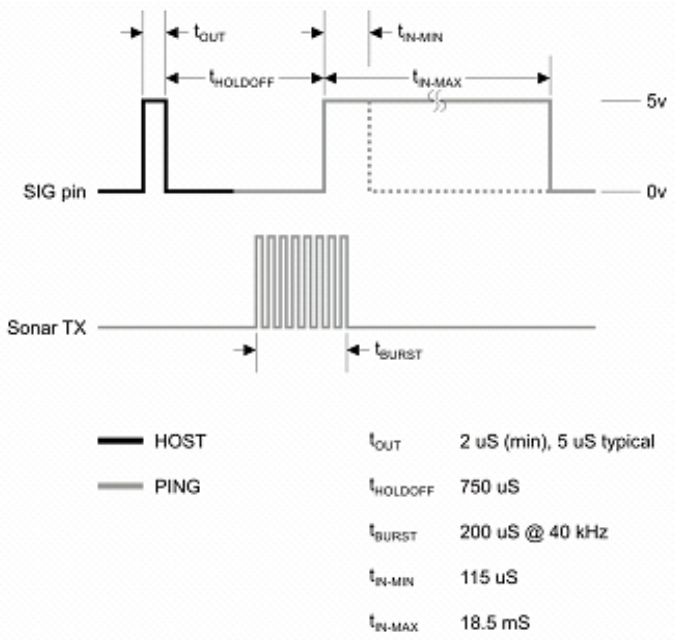
1. When plugging the cable onto the PING))) sensor, connect Black to GND, Red to 5 V, and White to SIG.
2. Check to see if your Board of Education servo ports have a jumper, as shown at right.
3. If your Board of Education servo ports have a jumper, set it to Vdd as shown.
4. If your Board of Education servo ports do not have a jumper, do not use them with the PING))) sensor. These ports only provide Vin, not Vdd, and this may damage your PING))) sensor. Go to the next step.
5. Connect the servo cable directly to the breadboard with a 3-pin header. Then, use jumper wires to connect Black to Vss, Red to Vdd, and White to I/O pin P15.



Board of Education Servo Port Jumper, Set to Vdd

Theory of Operation

The PING))) sensor detects objects by emitting a short ultrasonic burst and then "listening" for the echo. Under control of a host microcontroller (trigger pulse), the sensor emits a short 40 kHz (ultrasonic) burst. This burst travels through the air at about 1130 feet per second, hits an object and then bounces back to the sensor. The PING))) sensor provides an output pulse to the host that will terminate when the echo is detected, hence the width of this pulse corresponds to the distance to the target.

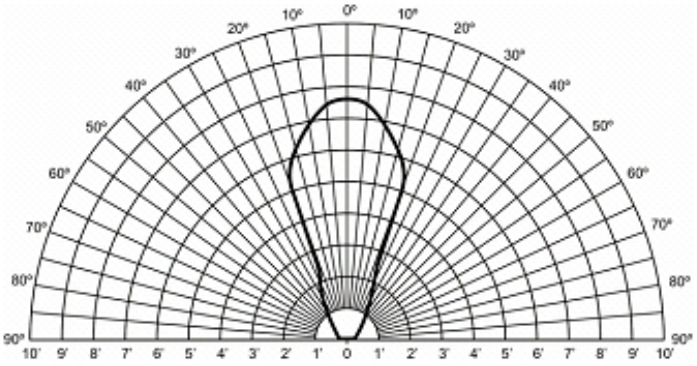


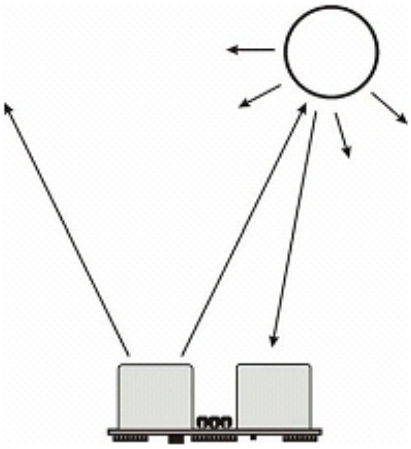
Test Data

The test data on the following pages is based on the PING))) sensor, tested in the Parallax lab, while connected to a BASIC Stamp microcontroller module. The test surface was a linoleum floor, so the sensor was elevated to minimize floor reflections in the data. All tests were conducted at room temperature, indoors, in a protected environment. The target was always centered at the same elevation as the PING))) sensor.

Test 1

Sensor Elevation: 40 in. (101.6 cm)
Target: 3.5 in. (8.9 cm) diameter cylinder, 4 ft. (121.9 cm) tall - vertical orientation





Program Example: BASIC Stamp 2 Microcontroller

The following program demonstrates the use of the PING))) sensor with the BASIC Stamp 2 microcontroller. Any model of BASIC Stamp 2 module will work with this program as conditional compilation techniques are used to make adjustments based on the module that is connected.

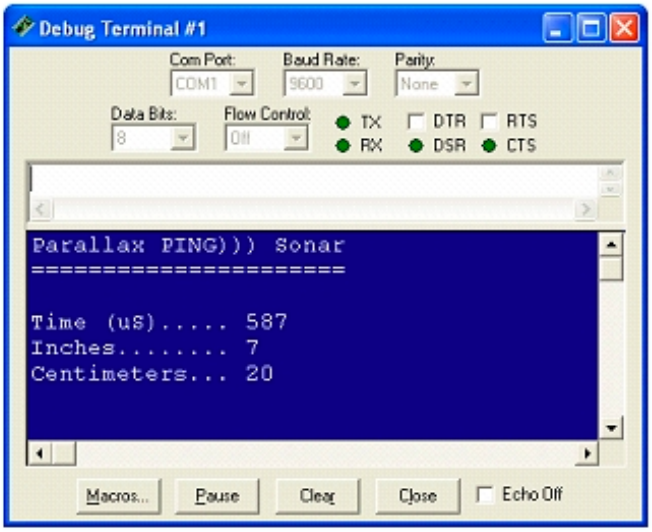
The heart of the program is the **Get_Sonar** subroutine. This routine starts by making the output bit of the selected IO pin zero - this will cause the successive **PULSOUT** to be low-high-low as required for triggering the PING))) sensor. After the trigger pulse falls the sensor will wait about 200 microseconds before transmitting the ultrasonic burst. This allows the BS2 to load and prepare the next instruction. That instruction, **PULSIN**, is used to measure the high-going pulse that corresponds to the distance to the target object.

The raw return value from **PULSIN** must be scaled due to resolution differences between the various members of the BS2 family. After the raw value is converted to microseconds, it is divided by two in order to remove the "return trip" of the echo pulse. The value now held in rawDist is the distance to the target in microseconds.

Conversion from microseconds to inches (or centimeters) is now a simple matter of math. The generally-accepted value for the speed-of-sound is 1130 feet per second. This works out to 13,560 inches per second or one inch in 73.746 microseconds. The question becomes, how do we divide our pulse measurement value by the floating-point number 73.746?

Another way to divide by 73.746 is to multiply by 0.01356. For new BASIC Stamp users this may seem a dilemma but in fact there is a special operator, **, that allows us to do just that. The ** operator has the affect of multiplying a value by units of 1/65,536. To find the parameter for ** then, we simply multiply 0.01356 by 65,536; the result is 888.668 (we'll round up to 889).

Conversion to centimeters uses the same process and the result of the program is shown below:



6. Tilt Sensor Datasheet

VTI TECHNOLOGIES

Data Sheet

THE SCA100T DUAL AXIS INCLINOMETER SERIES

The SCA100T Series is a 3D-MEMS-based dual axis inclinometer family that provides instrumentation grade performance for leveling applications. The measuring axes of the sensing elements are parallel to the mounting plane and orthogonal to each other. Low temperature dependency, high resolution and low noise, together a with robust sensing element design, make the SCA100T the ideal choice for leveling instruments. The VTI inclinometers are insensitive to vibration, due to their over damped sensing elements, and can withstand mechanical shocks of up to 20000 g.

Features

- Dual axis inclination measurement (X and Y)
- Measuring ranges ±30° SCA100T-D01 and ± 90° SCA100T-D02
- 0.0025° resolution (10 Hz BW, analog output)
- Sensing element controlled over damped frequency response (-3dB 18Hz)
- Robust design, high shock durability (20000g)
- High stability over temperature and time
- Single +5 V supply
- Ratiometric analog voltage outputs
- Digital SPI inclination and temperature output
- Comprehensive failure detection features
  - True self test by deflecting the sensing elements' proof mass by electrostatic force.
  - Continuous sensing element interconnection failure check.
  - Continuous memory parity check.
- RoHS compliant
- Compatible with Pb-free reflow solder process

Applications

- Platform leveling and stabilization
- 360° vertical orientation measurement
- Leveling instruments
- Construction levels

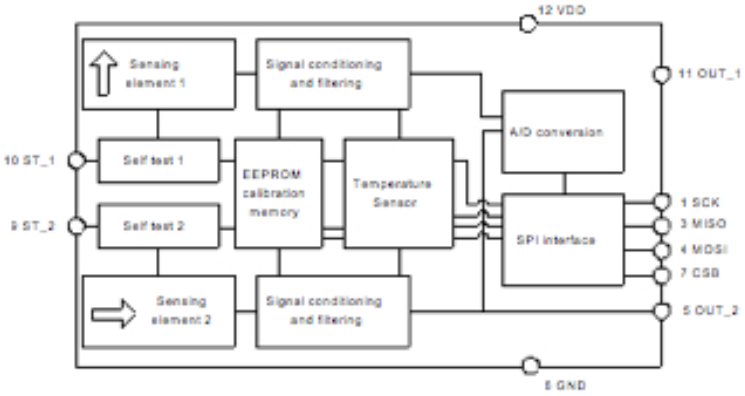


Figure 1. Functional block diagram

1 Electrical Specifications

The SCA100T product family comprises two versions, the SCA100T-D01 and the SCA100T-D02 that differ in measurement range. The product version specific performance specifications are listed in the table SCA100T performance characteristics below. All other specifications are common with both versions. Vdd=5.00V and ambient temperature unless otherwise specified.

1.1 Absolute Maximum Ratings

Supply voltage (VDD)	-0.3 V to +5.5V
Voltage at input / output pins	-0.3V to (VDD + 0.3V)
Storage temperature	-55°C to +125°C
Operating temperature	-40°C to +125°C
Mechanical shock	Drop from 1 meter onto a concrete surface (20000g). Powered or non-powered

1.2 Performance Characteristics

Parameter	Condition	SCA100T -D01	SCA100T -D02	Units
Measuring range	Nominal	±30	±90	°
		±0.5	±1.0	g
Frequency response	-3dB LP [1]	8-28	8-28	Hz
Offset (Output at 0g)	Ratiometric output	Vdd/2	Vdd/2	V
Offset calibration error		±0.11	±0.23	°
Offset Digital Output		1024	1024	LSB
Sensitivity		4	2	V/g
	between 0....1° [2]	70	35	mV/°
Sensitivity calibration error		±0.5	±0.5	%
Sensitivity Digital Output		1638	819	LSB / g
Offset temperature dependency	-25...85°C (typical)	±0.008	±0.008	°/°C
	-40....125°C (max)	±0.86	±0.86	°
Sensitivity temperature dependency	-25...85°C (typical)	±0.014	±0.014	%/°C
	-40....125°C (max)	-2.5...+1	-2.5...+1	%
Typical non-linearity	Measuring range	±0.11	±0.57	°
Digital output resolution		11	11	Bits
	between 0....1° [2]	0.035	0.07	° / LSB
Output noise density	From DC...100Hz	0.0008	0.0008	$° / \sqrt{Hz}$
Analog output resolution	Bandwidth 10 Hz [3]	0.0025	0.0025	°
Ratiometric error	Vdd = 4.75...5.25V	±1	±1	%
Cross-axis sensitivity	Max.	4	4	%
Long term Stability [4]		<0.014	<0.014	°

Note 1. The frequency response is determined by the sensing element's internal gas damping.

Note 2. The angle output has SIN curve relationship to voltage output refer to paragraph **Error! Reference source not found.**

Note 3. Resolution = Noise density * √(bandwidth)

Note 4. Power continuously connected (@ 23°C).

1.7 Electrical Connection

If the SPI interface is not used SCK (pin1), MISO (pin3), MOSI (pin4) and CSB (pin7) must be left floating. Self-test can be activated applying logic "1" (positive supply voltage level) to ST_1 or ST_2 pins (pins 10 or 9). Self-test must not be activated for both channels at the same time. If ST feature is not used pins 9 and 10 must be left floating or connected to GND. Inclination signals are provided from pins OUT_1 and OUT_2.

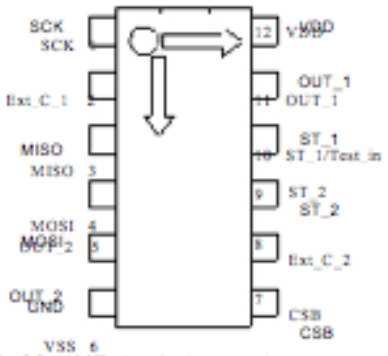


Figure 3. SCA100T electrical connection

No.	Node	I/O	Description
1	SCK	Input	Serial clock
2	NC	Input	No connect, left floating
3	MISO	Output	Master in slave out; data output
4	MOSI	Input	Master out slave in; data input
5	Out_2	Output	Y axis Output (Ch 2)
6	GND	Supply	Ground
7	CSB	Input	Chip select (active low)
8	NC	Input	No connect, left floating
9	ST_2	Input	Self test input for Ch 2
10	ST_1	Input	Self test input for Ch 1
11	Out_1	Output	X axis Output (Ch 1)
12	VDD	Supply	Positive supply voltage (+5V DC)

1.8 Typical Performance Characteristics

Typical offset and sensitivity temperature dependencies of the SCA100T are presented in following diagrams. These results represent the typical performance of SCA100T components. The mean value and 3 sigma limits (mean ± 3· standard deviation) and specification limits are presented in following diagrams. The 3 sigma limits represents 99.73% of the SCA100T population.

2 Functional Description

2.1 Measuring Directions

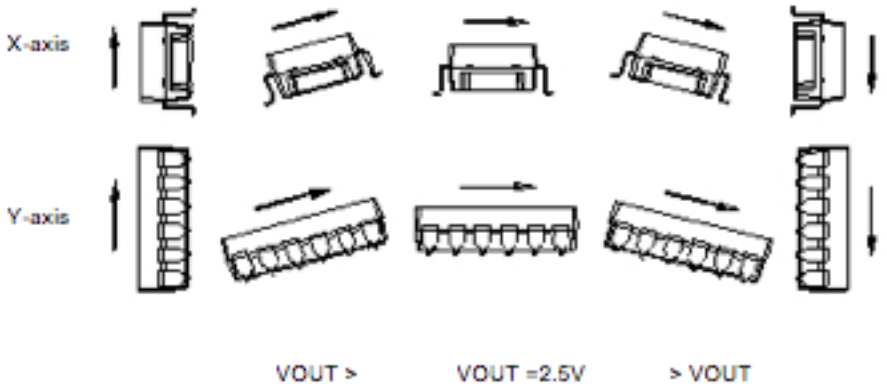


Figure 5. The measuring directions of the SCA100T

2.2 Voltage to Angle Conversion

Analog output can be transferred to angle using the following equation for conversion:

$$\alpha = \arcsin\left(\frac{V_{out} - Offset}{Sensitivity}\right)$$

where: Offset = output of the device at 0° inclination position, Sensitivity is the sensitivity of the device and V_{Dout} is the output of the SCA100T. The nominal offset is 2.5 V and the sensitivity is 4 V/g for the SCA100T-D01 and 2 V/g for the SCA100T-D02.

Angles close to 0° inclination can be estimated quite accurately with straight line conversion but for the best possible accuracy, arcsine conversion is recommended to be used. The following table shows the angle measurement error if straight line conversion is used.

Straight line conversion equation:

$$\alpha = \frac{V_{out} - Offset}{Sensitivity}$$

Where: Sensitivity = 70mV/° with SCA100T-D01 or Sensitivity= 35mV/° with SCA100T-D02

Tilt angle [°]	Straight line conversion error [°]
0	0
1	0.0027
2	0.0058
3	0.0094
4	0.0140
5	0.0198
10	0.0787
15	0.2185
30	1.668

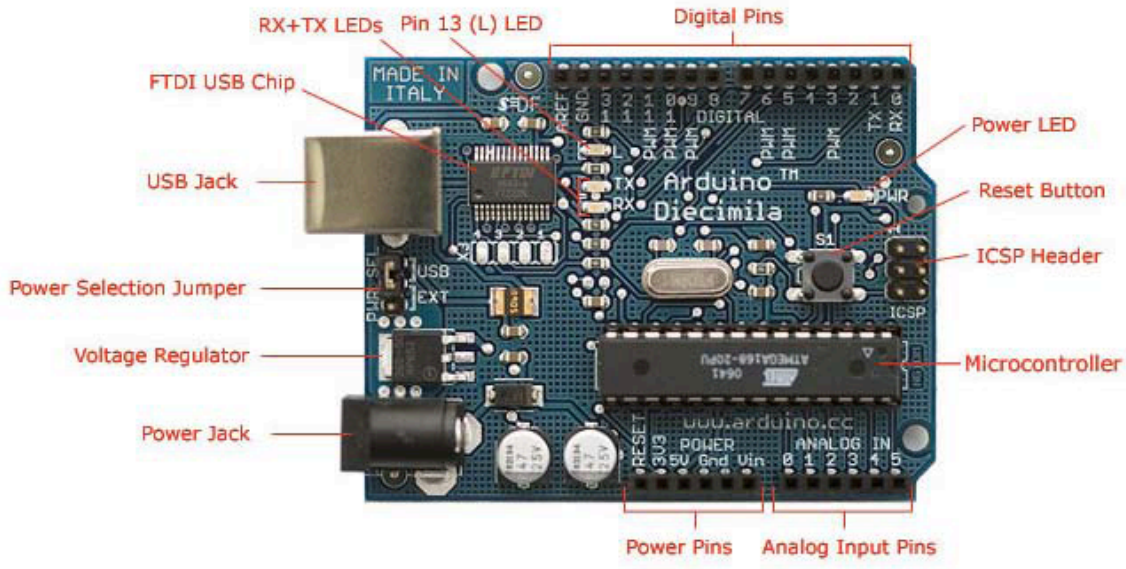
7. MatLab Simulation Code

   -Please see file BallandBeamSimulation.m submitted with the report.

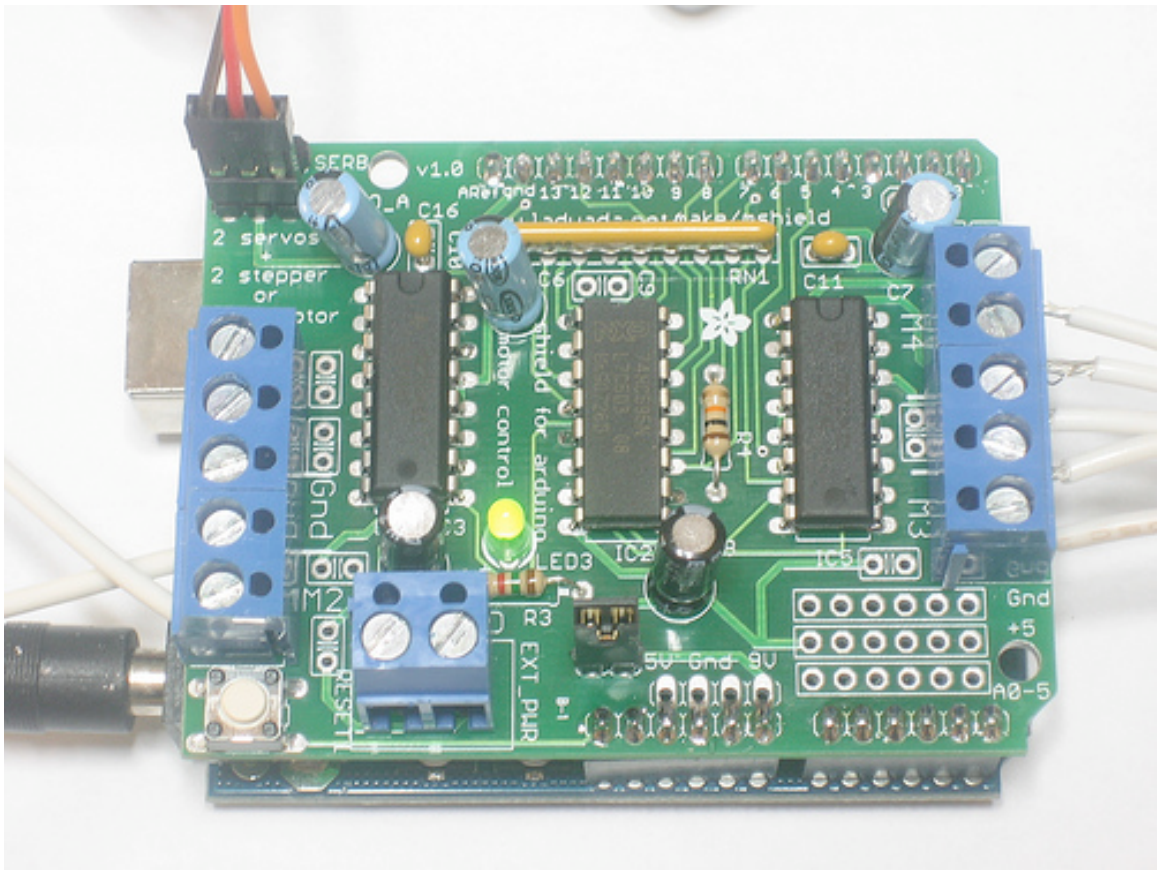
8. Arduino Control Code

   -Please see file Arduino_code.pdf submitted with the report.

9. Arduino and Motor Shield Pictures



Photograph by SparkFun Electronics. Used under the Creative Commons Attribution Share-Alike 3.0 license.


SERB
v1.0
2 servos
2 stepper
or
C16
C6
RN1
C11
C7
M4
M3
M2
Gnd
LED3
R3
IC2
IC5
EXT_PWR
5V Gnd 9V
Gnd
+5
A0-5
RESET
motor control
shield for arduino