DESIGN OF ROBOT MOTION PLANNING ALGORITHM FOR WALL FOLLOWING ROBOT

Oleh

Muhamad Khairul Bin Ali Hassan
(72488)

Disertasi ini dikemukakan kepada
UNIVERSITI SAINS MALAYSIA

Sebagai memenuhi syarat pengijazahan dengan kepujian dalam bidang Kejuruteraan Mekatronik bagi ijazah

SARJANA MUDA KEJURUTERAAN (KEJURUTERAAN MEKATRONIK)

Pusat Pengajian Kejuruteraan
Elektrik dan Elektronik
Universiti Sains Malaysia **Mei 2006**

ABSTRACT

This project demonstrates the use of Wall Following Algorithm for the development of mobile robot wall-following behaviors. Algorithms are developed for a simulated mobile robot that uses an array of range finders for navigation. Navigation algorithms are tested in a variety of differently shaped environments to encourage the development of robust solutions, and reduce the possibility of solutions based on memorization of a fixed set of movements. A typical wall-following robot evolutionary cycle is analyzed, and results are presented. Wall Following Algorithm is shown to be capable of producing robust wall-following navigation algorithms that perform well in each of the test environments used. In this project, i need to control the Robot to follow the wall. The robot is controlled by a BASIC Stamp microcontroller. The microcontroller is used to sequence the movement of the robot body sections via servos. The microcontroller also monitors the infrared sensor so that the ROBOT will avoid obstacles and wall as it explores. It can be automatically or manually control led through the internet/LAN. A user friendly GUI is developed using VISUAL BASIC to interface the user commands with the Robot controller. Computer A will be sent the data to computer B through the internet/LAN using JAVA program. Computer B will send the data to the BASIC Stamp and the BASIC Stamp will check their program. Then, the ROBOT will move depend on the program. The data will be sent to the ROBOT through the parallel port.

ABSTRAK

Bagi membangunkan projek ROBOT ini, ia dilaksanakan dengan menggunakan "Algoritma Menyusuli Dinding". Algoritma ini telah dibangunkan untuk simulasi robot dengan menggunakan susunan mencari jarak bagi mengemudikan Robot. Algoritma ini telah diuji dengan pelbagai bentuk keadaan untuk pembangunan penyelesaian Robot dan mengurangkan kemungkinan dalam penyelesaian untuk mengingat setiap pergerakan. Kebiasaannya, perkembangan peredaran "Robot Menyusuli Dinding" telah dianalisa dan keputusannya dipersembahkan. Algoritma Menyusuli Dinding berupaya mengemudikan Robot untuk menyusuli dinding dengan baik pada setiap suasana keadaan yang diuji. Dalam projek ini, saya perlu mengawal Robot untuk menyusuli dinding dengan sempurna. Robot ini dikawal dengan menggunakan pengawalmikro BASIC Stamp. Pengawalmikro ini digunakan untuk mengawal pergerakan berturutan badan robot yang disambung dengan servo motor. Pengawalmikro juga digunakan untuk memantau sinaran infra merah pada robot. Maka, ROBOT akan mengelak halangan atau dinding apabila infra merah mengesannya. Robot ini dapat dikawal secara automatik dan manual. Gambaran antaramuka pengguna (GUI) dibangunkan dengan menggunakan VISUAL BASIC untuk kegunaan pengguna bagi memberi arahan pengawalan ROBOT. Dengan menggunakan antaramuka VISUAL BASIC, komputer A akan menghantar data kepada komputer B melalui internet/LAN. Kemudian komputer B akan menghantar data yang diterima dan menterjemahkannya kepada BASIC Stamp untuk melaksanakan program yang dipilih. Kemudian ROBOT akan bergerak berdasarkan aturcara yang telah ditulis pada BASIC Stamp. Data daripada komputer B akan dihantar kepada ROBOT melalui port selari.

ACKNOWLEDGMENT

I would like to express my deepest appreciation to my supervisor, Dr G. Andal for her invaluable ideas and advice, particularly for discussion of design ideas and software relating to this project. I would also thank her for guidance and support throughout the course of the project.

Besides, I would like to thank technical staff in mechatronic lab for their guidance and help in using lab tools. My friends help who offered assistance in many and varied ways is greatly appreciated too.

Thank you.

You’re sincerely,

(Muhamad Khairul bin Ali Hassan)

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAM

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Real-time wall following and obstacle avoidance is one of the key issues to successful applications of mobile robot systems. All mobile robots feature some kind of collision avoidance, ranging from primitive algorithms that detect an obstacle and stop the robot in order to avoid a collision, through sophisticated algorithms, that enable the robot to detour obstacles. The latter algorithms are much more complex, since they involve not only the detection of wall and an obstacle, but also some kind of quantitative measurements concerning the wall and obstacle's dimensions. Once these have been determined, the wall following algorithm needs to steer the robot along the wall and resume motion toward the original target.

One approach to autonomous navigation is the wall-following method. Here the robot navigation is based on moving alongside walls at a predefined distance. If an obstacle is encountered, the robot regards the obstacle as just another wall, following the obstacle's contour until it may resume its original course. This kind of navigation is technologically less demanding, since one major problem of mobile robots the determination of their own position is largely facilitated. Naturally, robot navigation by the wall-following method is less versatile and is suitable only for very specific applications. One recently introduced commercial system uses this method on a floor cleaning robot for long hallways

Wall following was selected for the initial problem domain because it is a fairly simple problem to set up and evaluate. It also lays the groundwork for more complex problem domains, such as maze traversal, mapping, and full coverage navigation (i.e., vacuuming and lawn mowing). The development process for these behaviors is described, and the results of the experiments are presented.

1.2 OBJECTIVES

- Make the ROBOT move based on moving alongside walls by using the sensor.
- If an obstacle is encountered, the robot regards the obstacle as just another wall, following the obstacle's contour until it may resume its original course.
- Make the ROBOT move by manual control and automatic control through internet/LAN.
- To make the robot more intelligence to sense the wall and avoidance obstacles.
- Learn how to send the data from one computer to another computer through the internet/LAN.
- Make the microcontroller serve as the robot’s “brain” controlling and managing all functions, sensors, and reflexes

1.3 GANTT CHART

Project Schedule Chart

Task	Duration	JUN	JUL.	AUG.	SEPT	OCT.	NOV	DEC.	JAN.	FEB.	MAR	APR
1. Title of the project.	1 month	🟦	🟦									
2. Send proposal	1 month		🟦	🟦								
3. List of components	2 weeks			🟦	🟦							
4. Hardware design	2 month				🟦	🟦						
5. Find of components	2 month				🟦	🟦	🟦					
6. Software development	2 month						🟦	🟦				
7. Analysis of the ROBOT	3 month						🟦	🟦	🟦			
8. ROBOT development	5 month					🟦	🟦	🟦	🟦	🟦	🟦	
9. REPORT	4 month							🟦	🟦	🟦	🟦	
10. VIVA	2 weeks											🟦

1.4 Overview of the ROBOT Project

The robot that is built and programmed consists of a base, tire, R/C servo, battery pack, aluminum and electronics part. Here the robot navigation is based on moving alongside walls at a predefined distance. If an obstacle is encountered, the robot regards the obstacle as just another wall, following the obstacle's contour until it may resume its original course. Naturally, robot navigation by the wall-following method is less versatile and is suitable only for very specific applications. One recently introduced commercial system uses this method on a floor cleaning robot for long hallways (i.e., vacuuming and lawn mowing). The ROBOT is shown in **Figure 1** below

The robot is controlled by a BASIC Stamp microcontroller. The microcontroller is used to sequence the movement of the robot body sections via servos. The microcontroller also monitors an infrared sensor so that the ROBOT will avoid obstacles and wall as it explores.

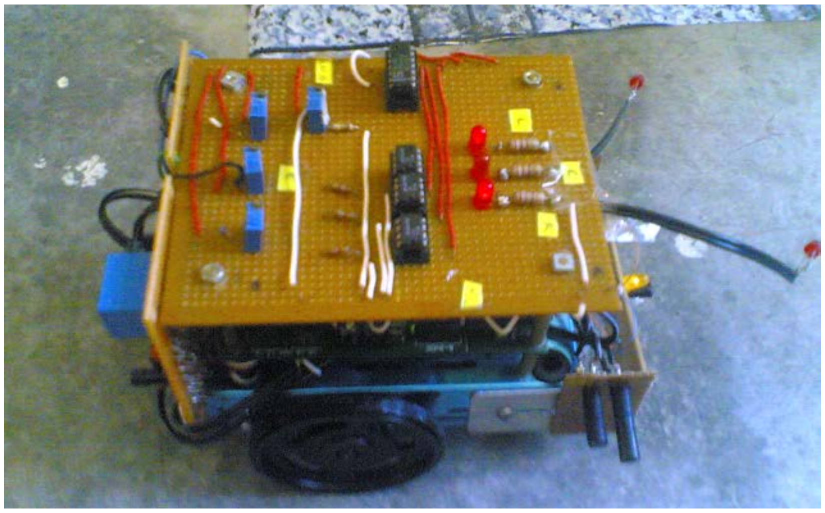


Figure 1: ROBOT

CHAPTER 2
MECHANICAL PARTS

2.1 Mechanical Construction of Robot

The construction of the robot will begin with the mechanical construction of the body. The parts needed for the mechanical construction are listed in **Table 1**

Table 1: List of Parts Required to build the ROBOT

Parts	**Quantity**
Part list for mechanical construction 1/16-inch thick aluminum stock	8feet x 10 feet piece
3/8inch machine screws	10
1/2 screws	10
Rod	4
Standard R/C servo and hardware	2
Battery pack	1
Standoffs	4
9/32” Rubber Grommets	2
13/32” Rubber Grommets	1

2.2 Constructing the Body Sections

Start by cutting a piece of the 1/16-inch aluminum to a size of 7-1/2 inches x 2-1/2 inches. These pieces will be identified as pieces A of the body sections. Use **Figure 2** as a guide to cut a piece to the dimensions shown.
File any rough edges from the pieces.

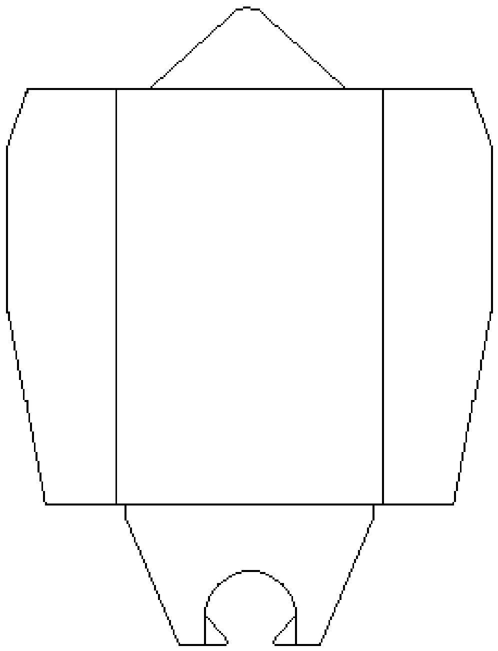


Figure 2: Cutting for ROBOT body.

When the piece is cut, use a 3/8-inch drill bit to drill the holes, as indicated in the diagram. A piece should look like the one pictured in **Figure 3**

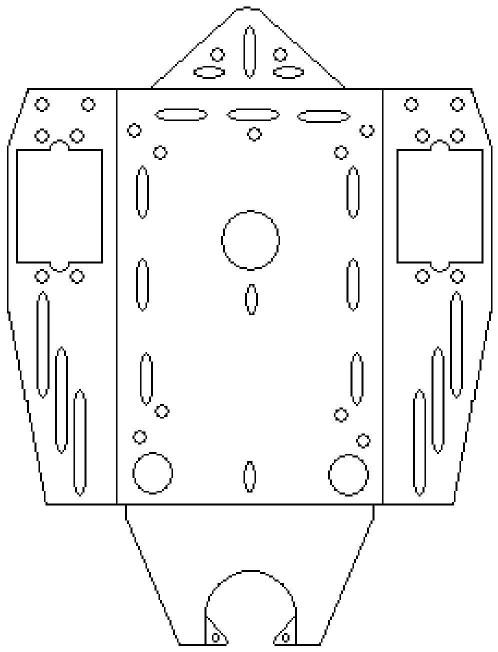


Figure 3: Bending and drilling guide for ROBOT body.

Bend each piece in a table vise/on the edge of a table, as indicated. A piece should look like the one pictured in **Figure 4a,4b** and **4c**

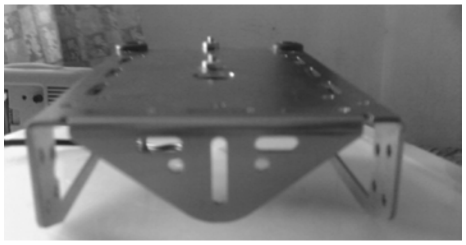


Figure 4a: Back view for ROBOT body

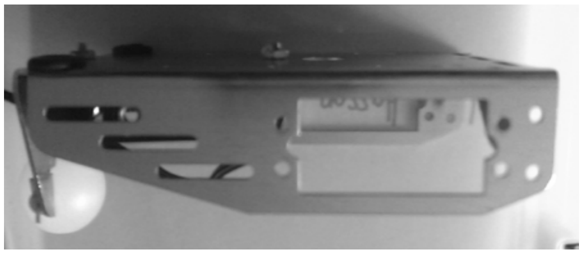


Figure 4b: Side view for ROBOT body

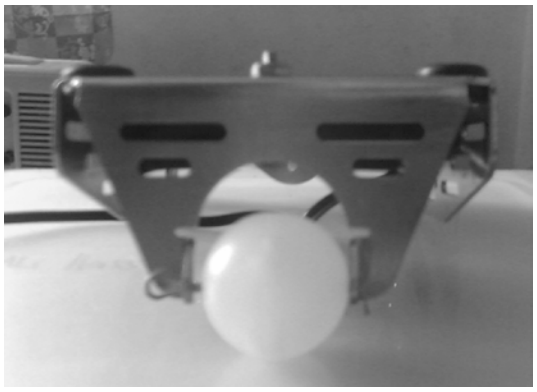


Figure 4c: Front view for ROBOT body

2.3 Assembling the Robot Structure

This section breaks assembling the Robot into steps. Now that all of the individual pieces that make up Robot mechanical body have been constructed, it is time to put them all together.

Parts List:

1) Body Section as shown in **Figure 5a**
2) Standoffs as shown in **Figure 5b**
3) ¼” 4-40 Screws as shown in **Figure 5c**
4) 9/32” Rubber Grommets
5) 13/32” Rubber Grommets

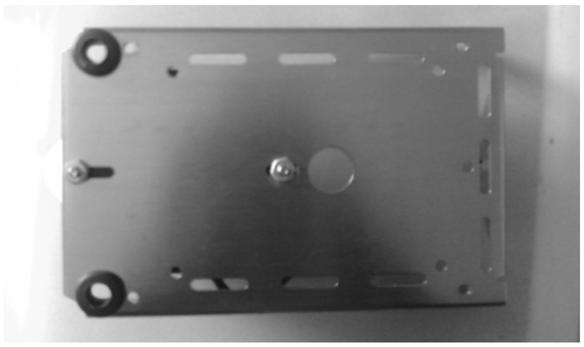


Figure 5a

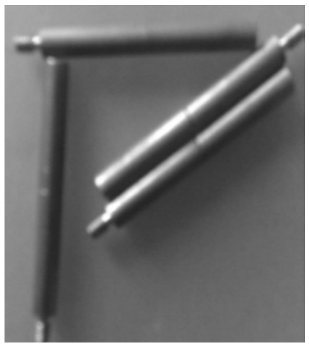


Figure 5b

Figure 5c

- Start by connecting the 13/32” rubber grommet into in the center of the body.
- Insert the two 9/32” rubber grommets into corner holes as shown.
- Use four ¼” 4-40 screws to attach the four standoffs to the body as shown in Figure below.

2.4 Servo Motor

Get the two servos as shown in **Figure 6**. The servo must be modifying first.

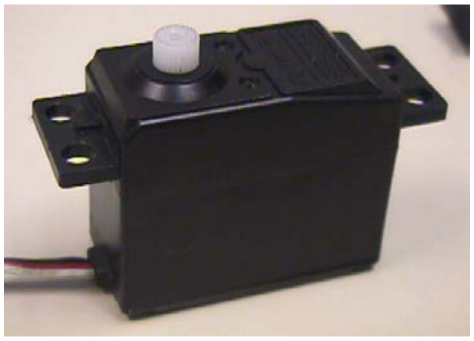


Figure 6: Servo Motor

This is the step to modify the servo as shown in **Figure 7** below. Unscrew each of the screws, then pull each components part by part as shown in figure below. Servo must be modification for a fully-rotational Servo (360°)

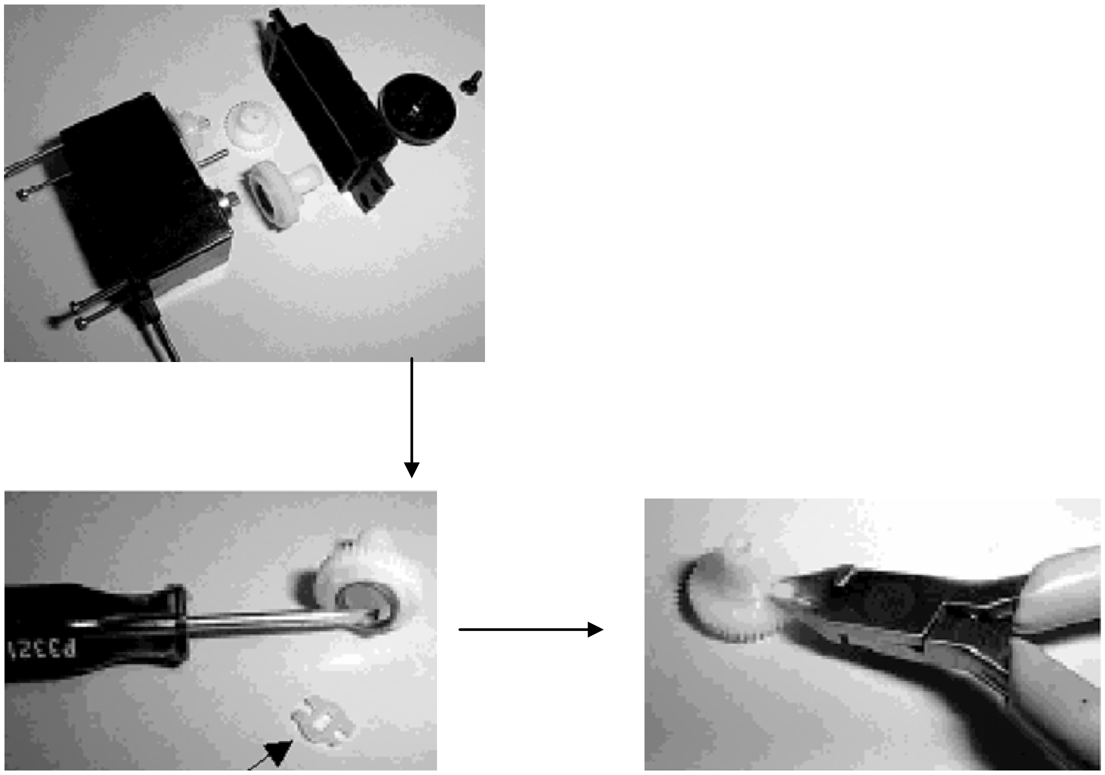


Figure 7: Step to modify the servo

Now, use the eight 3/8” 4-40screws and locknuts to attach each servo to the Body of the Robot. Also plug the battery pack back into the body of the Robot as shown in **Figure 8**. Use flathead screw and locknuts to attach the battery pack to underside of the robot body. Make sure to insert the screw through the battery pack then tighten down the locknuts on the topside of the chassis. As shown in figure below.

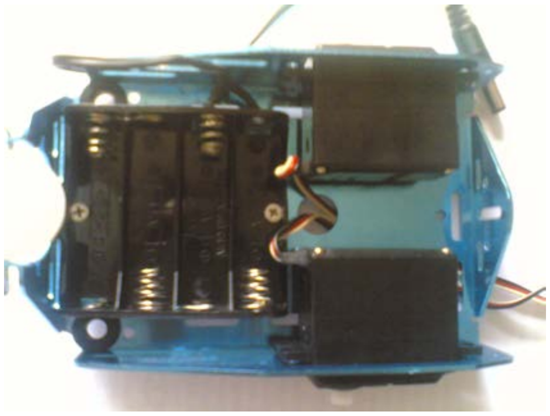


Figure 8: Battery pack and Servo installed.

2.5 Socketing the Basic Stamp 2.

Figure 9 shows the Basic Stamps attached to the Robot chassis with the servo plugged into the servo ports. Make sure the white breadboard on the board is above where the servos are mounted on the chassis. Use the four ¼” machine screws to attach the board to the standoffs.

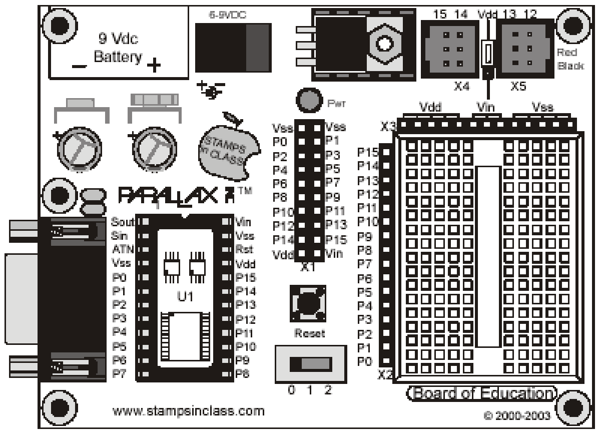


Figure 9: BASIC Stamp2

The numbers along the top indicate the servo port number. If we connect a servo to the servo port 12, it means the servo's control line is connected to I/O line P12. I/O line P12 is a metal trace on the board that connected the top servo port pin to the BASIC Stamp's I/O pin P12.

2.6 The Wheels

Figure 10a shows the wheel attached to the Robot Chassis with a cotter pin and **Figure 10b** shows one of the front wheels attached to a servo's output shaft.

The plastic ball used as the Robot rear, and cotter pin is its axle. Each plastic wheel has a recess that fits on a servo output shaft. Press each plastic wheel onto a servo output shaft making sure what shaft lines up with and sinks into the recess.

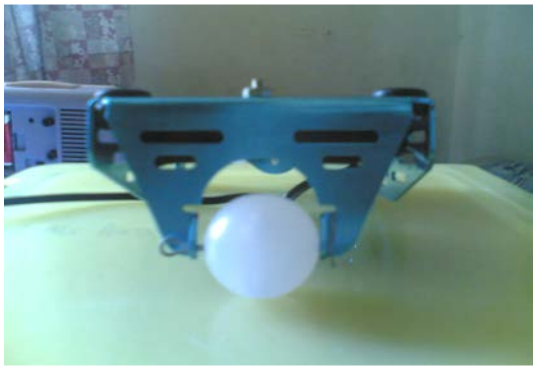


Figure 10a: Wheel attached to the Robot Chassis with a cotter pin

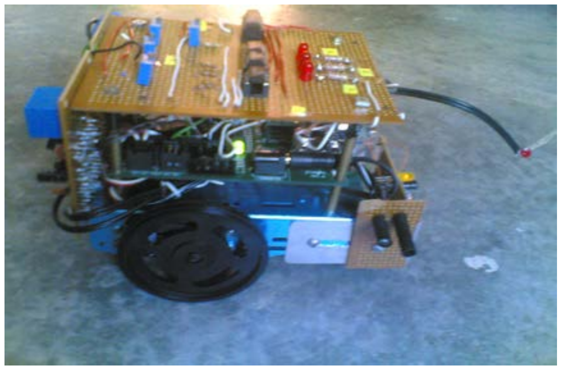


Figure 10b: one of the front wheels attached to a servo's output shaft

CHAPTER 3

TESTING THE SERVOS INDIVIDUALLY

3.0 TESTING THE SERVOS INDIVIDUALLY

3.1 How the servos work

Normally, these servos are designed to control the position of something such as a steering flap on radio-controlled airplane. Their range of motion is typically 90 or 180, and they are great for applications where inexpensive, accurate high-torque positioning motion is required. The position of these servos is controlled by an electronic signal called a pulse train, which you'll get some first hand experience with shortly. An un-modified hobby servo has build-in mechanical stoppers to prevent it from turning beyond its 90 or180 range of motion. It also has internal mechanical linkages for position feedback so that the electronic circuit that controls the Dc motor inside the servo knows where to turn to in response to a pulse train.

Figure 11a shows the circuit that is established when a servo is plugged into the servo port labeled 12 on the board. The red and black wires connect to the servo's power source and the white (or sometimes yellow) wire is connected to a signal source. When a servo is plugged into servo port 12, the servo's signal is BASIC Stamp I/O pin P12.

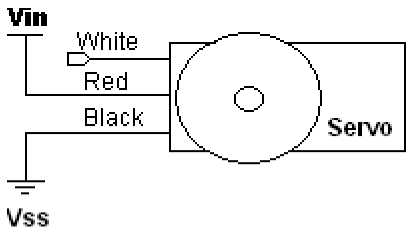
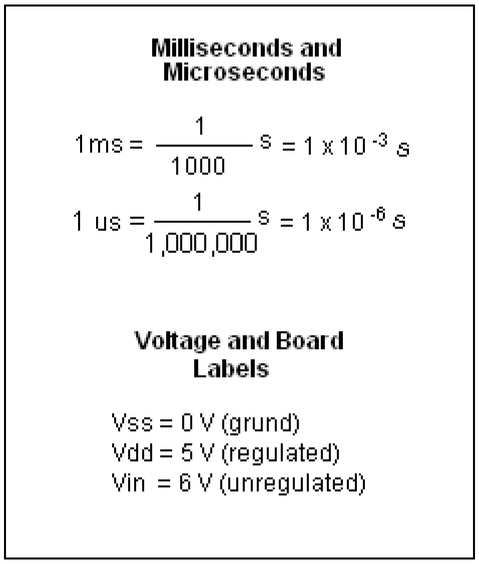


Figure 11a: Servo Circuit

Amount of time will be reoffered to in units of seconds (s), milliseconds (ms) and microseconds (us). Seconds are abbreviated with the lower-case letter s. So,,one second is written as 1 s. Milliseconds are abbreviated as ms and it means one one-thousandth of a second. The milliseconds and Microseconds box below shows these equalities in terms of both fractions and scientific notation.

A voltage level is measured in volts, which is abbreviated with an upper case V. The Board has sockets labeled Vss,Vdd and Vin as shown in **Table 2** below.. Vss is called the system ground or reference voltage. When the battery pack is plugged in, Vss is connected to its negative terminal. As far as the Board, BASIC Stamp and serial connections to the computer are concerned, Vss is always 0 V. Vin is unregulated 6 V, and it's connected to the positive terminal of the battery pack. Vdd is regulated to 5 V by the Board onboard voltage regulator, and it will be used with Vss to supply power to circuits built on the Board breadboard.

Table 2 : Voltage and Board Labels



Milliseconds and Microseconds

$$1\,\text{ms} = \frac{1}{1000}\,\text{s} = 1 \times 10^{-3}\,\text{s}$$

$$1\,\text{us} = \frac{1}{1{,}000{,}000}\,\text{s} = 1 \times 10^{-6}\,\text{s}$$

Voltage and Board Labels

Vss = 0 V (grund)
Vdd = 5 V (regulated)
Vin = 6 V (unregulated)

The control signal the BASIC Stamp sends to the servo's control line is called "pulse train" and an example of one is shown in **Figure 11b** The BASIC Stamp can be programmed to produce this waveform using any of its I/O pins. In this activity, we'll start with I/O pin P12, which is already connected to servo port 12 by a metal trace built into the Board. First, the BASIC Stamp sets the voltage at P12 to 0 V (low) for 20ms. Then\, it sets the voltage at P12 to 5V (high) for 1.0 ms. Then, it starts over with a low output for another 20 ms, and high output for another 1.0ms, and so on.

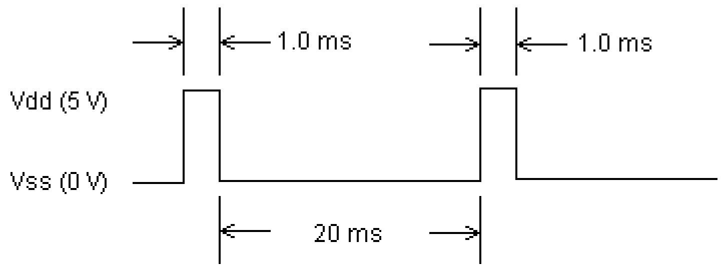


Figure 11b: Pulse Train

This pulse train has 1.0ms high time and a 20ms low time. The high time is the main ingredient for controlling a servo's motion, and it is most commonly referred to as the pulse width. Since these pulse go from low to high (0 V to 5 V) fro a certain amount of time, they are called positive pulse. Negative pulses would involve a resting state that's high with pulses that drop low. Pulse trains have some other technical descriptions duty and duty cycle. These are described in BASIC Analog and Digital.

A pre-modified servo can be pulsed to make its output shaft turn continuously. The pulse widths for pre-modified servos range between 1.0 and 2.0ms for full speed clockwise and counterclockwise respectively. If we give ser 1.25ms pulses, it will turn clockwise at rough half of full speed. If we give servo 1.90 ms pulses, the servo will turn at almost full speed counterclockwise. The "center pulse width" is 1.5 ms, and that makes the servo stay still

3.2 Calibration of the Servos in Software

The servos can adjust a pulsout command's period argument to straighten out the Robot travel. This practice is called "calibration in software. If the Robot veers to the right when is programmed to go straight forward, either the left wheel needs to slow down, or the right wheel needs to speed up. Since the servos are pretty close to top speed as it is, slowing the left wheel down will work better. You can do this by making the pulse period to the left servo, which connected to P13, smaller. For example, instead of using the command pulsout 13, 1000, we might try pulsout 13,900. Keep in mind that the adjustment is different if we need to slow the right wheel's rotation. In that case, use a pulsout period argument larger than 500, such as 560 for starters. By trying different values, we can home in on the value that will get the Robot wheels turning forward at the same speed, then the Robot will move forward in the straight line.

We can also make the same corrections to get the Robot travel in a straight line backward. We must modify Program to make the Robot travel full speed backward. Then, test to figure out which wheel needs to slow down. Since the pulsout period values have to be swapped to make the Robot travel full speed reverse, we will also need to adjust the pulsout period argument differently. For a given wheel, slowing it down involves taking the full speed pulsout period argument, and adjusting it so that is slightly closer to the center pulsout period of 750.

- Make the necessary changes in Program so that it makes the Robot go full speed backward.
- Make the additional adjustment to one of the two pulsout commands to slow down the wheel that’s turning too fast to cause the Robot to travel backwards in a straight line.
- Make notes of the fine-tuned pulsout period values you came up with for full speed straight forward and full speed straight backward.

This calibration involves testing each servo at pulse periods around the predicted center pulsout period value of 750(1.5ms). What we will be looking for s the best period value to really make the servo stay still.

First, run Program. If the servos don’t move, they are calibrated. If the servos rotate slowly, follow the instruction below. If the servos turn rapidly, check the program for clerical errors. Program below is using to check the servo.

```
'{$STAMP BS2}    'Stamp directive

LOW 12          'set P12 to output-low
LOW 13          'ser P13 to output-low

loop:
 PULSOUT 12,750 'sent 1.5ms pulses to P12
 PULSOUT 13,750 'sent 1.5ms pulses to P13
PAUSE 20            'every 20ms

GOTO loop         'sent program to "loop:" label
```

By viewing the slowly turning Robot wheel from the side, you can decide whether to search for the true center *pulsout period* using values above or below 750.

- If the wheel is turning clockwise, that means that its true center pullout period is somewhere above 750.
- If the wheel is turning counterclockwise, it means the true center *pulsout period* is somewhere below 750.

The test for finding the correct center pulse width for each servo involves finding the range of values that make the servo stay still. If the servo is turning slowly clockwise, try pulsout period values of 751,752,753…To figure out where are true center is, we will need to keep on increasing the pulsout period argument. Make a note of the first value that causes the servo to stay still. Then, keep increasing the pulsout period until the servo starts turning the opposite direction. Make a note of that value as well. Keep in mind that this process works almost the same fro a servo that is turning counterclockwise, simply look for a true center pulse width somewhere below 750. Your search will begin at 749, then continue downwards until the servo stops turning, then starts turning the opposite direction.

After finding the upper and lower pulsout period values that na\make your servo stay still, you can take the average of those two values the true center pulsout period argument. This calculated value is the one you should use in place of 750 in Program Listing.

Here’s an example that’s a little more difficult. What if the servo doesn’t stop until it gets a period argument of 752? Then, what if it doesn’t start turning again until gets a period argument of 757? The true period to center this modified servo is would be somewhere between 752 and 757, but where? By taking the average of the two numbers, the period would be:

$$\boxed{\text{average pulsout period} = \frac{752 + 757}{2} = 754.5}$$

So, 754.5 can not be used as a valid period argument. The average pulsout period needs to be rounded, but which way? Although the standard practice is to round up if the decimal value is 0.5 or above, the servo might not really behave according to that practice.

CHAPTER 4
SOFTWARE

4.0 SOFTWARE

4.1 Introduction of Basic Stamp II

The BASIC Stamp is a microcontroller developed by Parallax, Inc. which is easily programmed using a form of the BASIC programming language. It is called a “Stamp” simply because it is close to the size of an average postage stamp, except for the BS2p40 which is much longer due to it's additional I/O pins as shown in **Figure 12a.**

Basic Stamp II will be employed throughout the lab. Basic Stamp II (from Parallax) is a microcontroller module based on Microchip PIC16C57. Basic Stamp II (BS2) module NX-1000

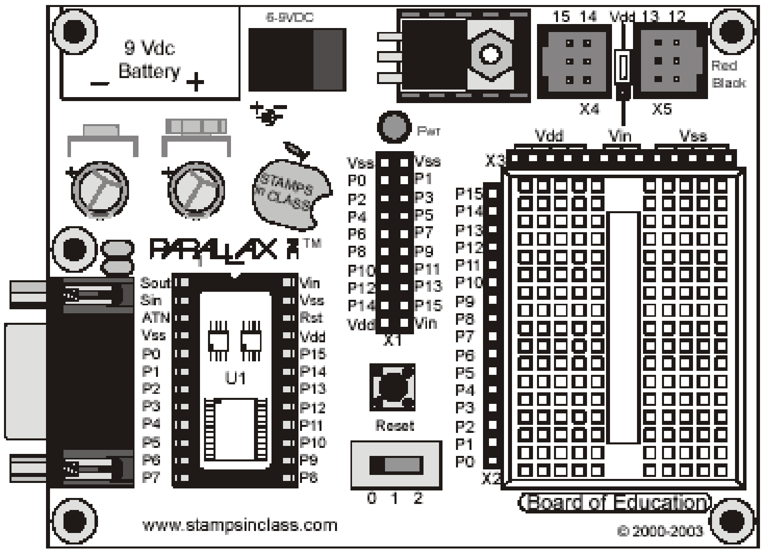


Figure 12a: Basic Stamp II.

4.1.1 Using the BASIC Stamp Breadboard

The breadboard has many strips of copper which run underneath the board in a horizontal fashion. These strips connect the sockets to each other. This makes it easy to connect components together to build circuits. To use the breadboard, the legs of components or wires are placed in the sockets. The sockets are made so that they will hold the component in place. Each hole is connected to one of the metal strips running underneath the board. Each metal strip forms a node. A node is a point in a circuit where two components are connected. Connections between different components are formed by putting their legs in a common node. There are two columns of 17 nodes on the breadboard. Each node contains five holes as shown in **Figure 12b.**

For chips with many legs (ICs), place them in the middle of the board so that half of the legs are on the left side and half are on the right side. Nodes on the left side are not connected to nodes on the right side.

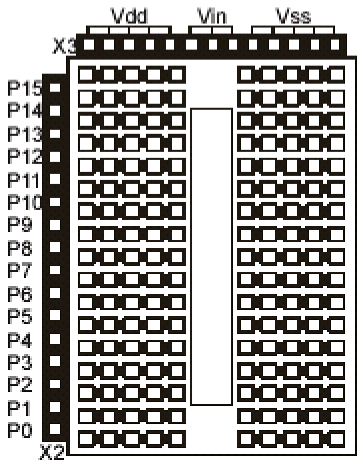


Figure 12b: Breadboard

4.1.2 Example Circuit

On the left is a simple circuit used to monitor light levels. The illustration on the right shows how this circuit (Figure **13a**) can be constructed on the breadboard as shown in **Figure 13b**

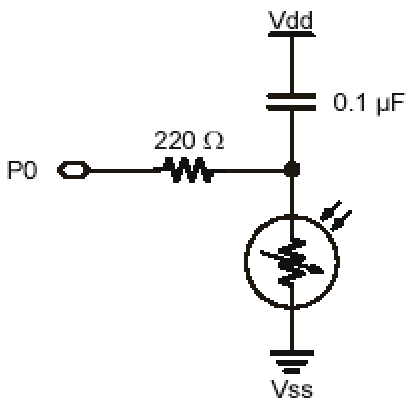


Figure 13a: Circuit

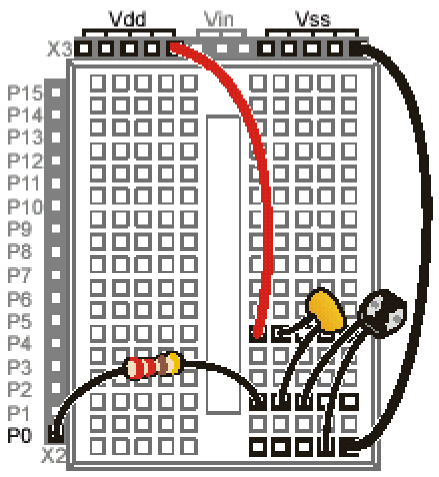


Figure 13b: Constructed on the breadboard