David Pinto
EEL5666 – Fall 2001

Run-N-Shoot

TABLE OF CONTENTS

ABSTRACT

This paper focuses on the design and implementation of an autonomous agent designed to locate a ball and shoot it at a target.

INTRODUCTION

The objective of this project was to better understand the issues involved in constructing an autonomous agent and gain experience in developing real-time image-processing software. In order to accomplish this task, a robot with the following design goals was implemented:

- Locate a selected object from the screen
- Acquire the object
- Locate and line-up with a second selected object
- Propel the acquired object towards the second object

INTEGRATED SYSTEM

The agent is controlled by a wireless link to a desktop computer. The desktop computer processes the video from the robot and sends feedback to the robot. The following is a layout of the system.

MOBILE PLATFORM

The platform was built on the fly from 1/4" plywood. The frame was modified as needed to satisfy the demands of the project as they c ame up. The lifeline of the platform is as follows:

1. A TJ-like platform was put together – a box to mount the motors and house the processor & batteries along with a circular top to mount the camera onto.

2. A platform was appended to the front of the box with a channel in the center so that the ball could be captured.

3. A crescent-moon-like section of the circular top was cut away so that the camera could have a view of the newly appended platform.

4. Various holes were cut away to allow for tie-wraps and wire-passage.

ACTUATION

There are four servos controlled by the microprocessor. Two servos are used for displacing the robot, one for locking the ball in position and one for shooting the ball. The following is a detailed explanation of each servo and its function.

- Servo 1 & 2: These servos are hacked Hi-Tec HS605-MGs. By hacking the servos full rotation of the wheels was permitted. These servos provided the means by which the robot moved around.

- Servo 3: This servo is an un-hacked Hi-Tec HS300. The servo has a piece of wood resembling a paddle glued on to the end of it and "locks" the ball in the channel when the ball is in the desired position. It does this by rotating 90-120 degrees at the appropriate time.

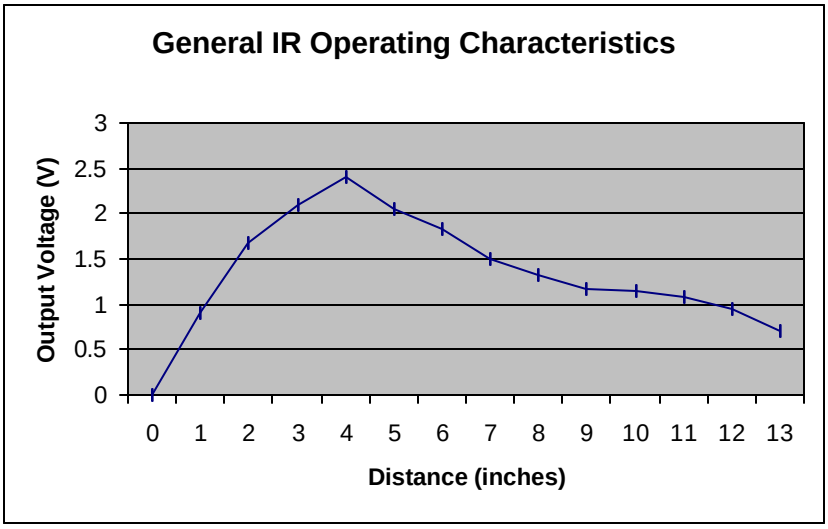
- Servo 4: This servo is an un-hacked Hi-Tec HS300. The servo has a spring screwed onto the end of it and "whips" the ball in order to shoot it when desired.

SENSORS

Three types of sensors were mounted and calibrated on the robot, but only two were used. IR sensors, bump sensors and video were the three sensors that were integrated, but only the video and the bump sensors were used. The IR sensors did not facilitate the robot's objectives. The following is a more detailed explanation of each sensor and their function.

IR Sensors

The IR sensors used were 4 SHARP GP2D12 14s. The IRs were mounted on the front and the side of the robot to detect walls/obstacles. The sensor interface consisted of a 5V source, a ground, and a 0-2.5V output. The output was directly connected to the HC11's A/D system. The following graph is a generalized characteristic of their operation.



Bump Sensors

Four bump-sensors were used to detect when the robot was in contact with a wall. The sensors used were 2 Radio Shack SPDT (275-016) switches and 2 Radio Shack SPDT Switches With ¾" Roller Lever (275-017). The switches without the rollers were placed on the front of the robot to detect when a wall was frontally contacted. The switches with the rollers were placed on the left and right front ends of the robot to detect when the robot was in contact with a wall on its left or right side – this helped decide whether the robot should turn left or right. The sensors were organized in groups of 2 on each side of the robot – in each group 5V indicated frontal contact, and 2.5V indicated side contact.

Video Sensor

The video sensor is implemented by using a wireless X10 camera and a desktop computer running the JRE (Java Runtime Environment 1.3). All software was implemented using

Java. The object detection algorithm works by determining the mean of the pixels that lie within some RGB threshold and taking action based on that mean. The threshold is determined by the user during runtime and the actions include turning left, turning right, going forward, shooting the ball, and capturing the ball. To determine the RGB threshold the user clicks on the desired object and enters a number signifying the number of standard deviations away that the thresholds should be set at. The mean and variance of the selected area is calculated and the threshold is set accordingly. During runtime, the mean of the pixel positions that were within the threshold region is calculated and action is taken based on the mean. The bounds for the action taken is determined by the user during runtime as well. The user defines lines signifying the left bound, right bound and line of sight for the target. Additionally a region is selected for the "capture" region.

BEHAVIORS

Avoiding Walls/Obstacles

This behavior makes it so that the robot does not get stuck on a wall. The bump sensors indicate that the wall has been hit. The robot then backs up for 1 -2 sec and then turns in the appropriate direction to avoid the wall – the direction is chosen as depicted by the bump sensors. If the sensors have no information to provide, then a random direction is chosen.

Finding a Ball

In this state the robot is looking for a ball. If the ball in not in the camera's view, the robot turns to the right until the ball is in view. It then targets the ball and moves forward until the ball in position to "capture" the ball.

Finding a Target

After the ball has been captured, the robot goes into this mode. If the target is not in view of the camera, the robot turns to the right until it is in view. The robot then lines up with the target and goes into shooting mode.

Capturing a Ball

When the ball is in a certain region, a lever is toggled to go down to capture the ball.

Shooting a Ball

When the robot is lined up with the target, the robot goes into shooting mode and lifts the "ball-capture" lever and shoots the ball.

TECHICAL INFORMATION

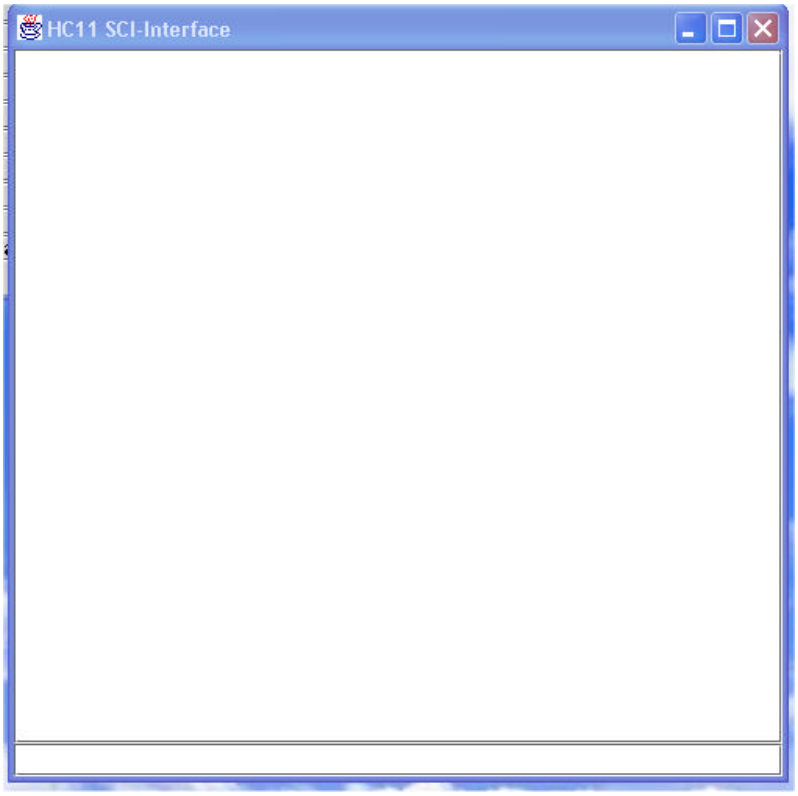
Communications Interface

In order to communicate to the robot a wireless interface was implemented. In order to overcome noise issues a communications protocol was created.

The desktop sends commands as an 8-bit command. The processor interprets the command and executes the command for about 30ms. This way if an unintended command was interpreted, it doesn't go crazy forever but only for 30ms. After the command is processed a "ready" signal is sent back to the desktop letting it know that the command has been processed and it is ready for more. If the desktop does not receive a "ready" command within 3 seconds, it polls the robot asking if it ready until it receives a ready command.

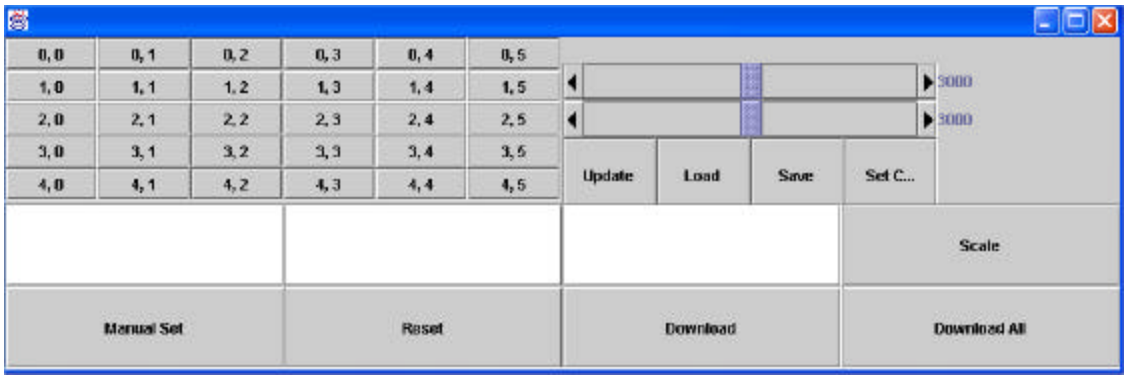
Additionally a SCI interface that the user can interface with the robot was made. The following is a snapshot of it.

The *Connect* and *Disconnect* buttons enable/disables feedback to the robot.



Kinematics Interface

In order to calibrate the servos and set the "directional" controls, a kinematics control interface was implemented. The following figure is a screenshot of the interface.



The sliding bars on the right represent each servo. The center is the "idle" position. As the bars slide the servos increase and decrease their respective speeds.

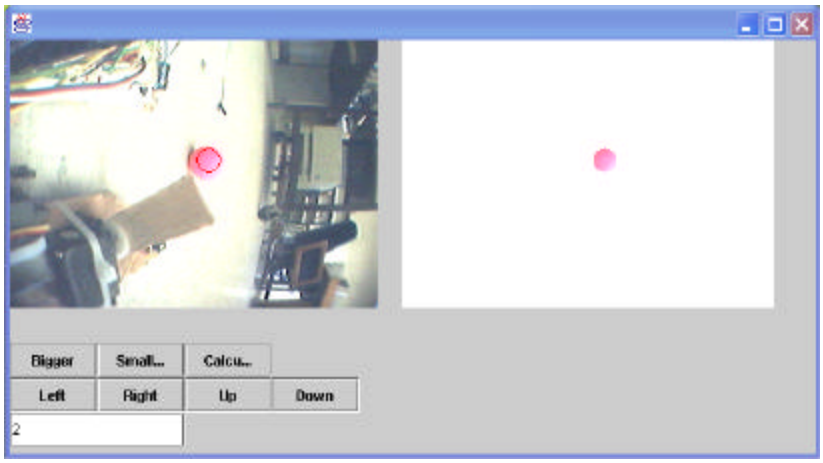
The grid with the coordinates on it represent the programmable directions. To program a direction, a coordinate is clicked, the desired servo settings are set and then *Update* button is clicked which saves the setting. Additionally, the user can type the servo values manually in the textboxes provided on the bottom and clicking on the *Manual Set* icon. The *Reset* button sets the selected direction to idle. A set of directions can be saved by clicking the *Save* button and loaded by clicking the *Load* button. To set the "idle" values an idle position is set with the scrollbar.

The *Scale* button scales all the speeds by the factor provided by the user.

The *Download* button downloads the selected sped to the robot. The *Download All* button downloads all the setting to the robot.

Object Recognition Interface

In order to select the object/target the following interface was implemented.

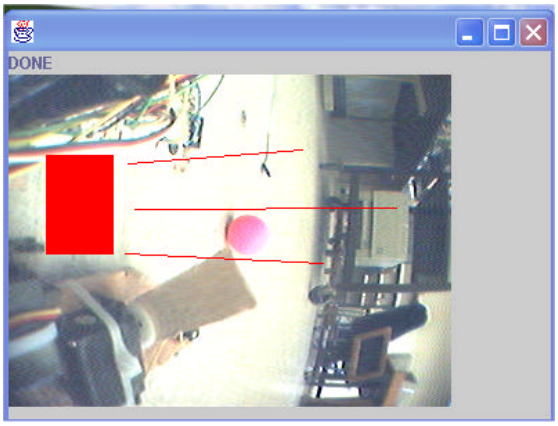


The desired object is clicked on. To more precisely vary the size and position of the circle the *Bigger*, *Smaller*, *Left*, *Right*, *Up* & *Down* buttons are provided. After the object is encapsulated, the user e nters the number of standards deviations away from the mean that the threshold should be set at by entering the number in the textbox and then clicking on the *Calculate Histogram* button.

Bounds Interface

To dynamically set the bounds of the feedback bounds the interface on the right was implemented.

The shaded region is the region where the ball is “captured”. The line in the center is the



CONCLUSION

My objective was reached and I feel that I completed what it was I set out to do. The robot is not perfect in its design and it would be nice to implement more advanced image-processing algorithms and make the hardware for displacing the robot more precise. Additionally, it would be nice to make another robot, have them communicate and play a "soccer" game.

HC11 Code

```
#include <hc11.h>
#include <mil.h>
#include <stdio.h>

/*Interrupt Procedures/Variables*/

#pragma interrupt_handler isrTOC2
#pragma interrupt_handler isrTOC3
#pragma interrupt_handler isrTOC4
#pragma interrupt_handler isrTOC5
#pragma interrupt_handler rtii

#define LEFT_BUMPER_THRESHOLD 90
#define RIGHT_BUMPER_THRESHOLD 90
#define leftIR ADR3
#define rightIR ADR1
#define rightBumper ADR4
#define leftBumper ADR3
#define gateUp 4000
#define gateDown 5000
#define shootWidth 500
#define stopSh ootWidth 4500

  int debug = 1;
  int menuDisplay = 1;

  int period = 40000;

  int pulseWidth_TOC3 = 3000;
  int pulseWidth_TOC2 = 3000;
  int pulseWidth_TOC4 = 3000;
  int pulseWidth_TOC5 = 3000;

  int pulseOn_TOC2 = 0;
  int pulseOn_TOC3 = 0;
  int pulseOn_TOC5 = 0;
  int pulseOn_TOC4 = 0;

  int rtiInterrupts = 0;
  int rtiTime = 0;

  int shooting = 0;
  int ADStatus = 0;
  int sensorCheckEnable = 0;
  int wirelessMode = 0;

  int speed[5][6][2];

/*Prototypes*/

  void processSelection(int);
  void displayMenu(void);
  int parseInputToInt(void);
  int readAD(void);
  void checkSensors(void);
  void setSpeed(int,int);
  void shootSolenoid(void);
  void initializeSystem(void);
  void initializeServos(void);


/*MAIN*/
  void main(void)
  {
    char menuSelection = '0';

    initializeSystem();
```

```
    while (menuSelection != 9)
    {
      displayMenu();
      menuSelection = ( (int)getchar() );
      processSelection(menuSelection);
    }

    printf("System. off.\n");
  }

  void initializeSystem(void)
  {
    int i = 0;
    int j = 0;

    printf("Initializing System.\n");
    setbaud(BAUD9600);
    initializeServos();

    rtiInterrupts = 0;
    rtiTime= 90;

    shooting = 0;
    CLEAR_BIT(PORTA, 0x10);
    ADStatus = 0;
    CLEAR_BIT(OPTION, 0x80);
    sensorCheckEnable = 0;
    wirelessMode = 0;

    pulseWidth_TOC2 = 3000;
    pulseWidth_TOC3 = 3000;
    pulseWidth_TOC5 = gateUp;
    pulseWidth_TOC4 = stopShootWidth;

    TCTL1 = 0;



    init_serial();
    printf("Done Initializing.\n");
  }

/*Present Menu*/
  void displayMenu(void)
  {
    if (menuDisplay)
    {
      printf("\n\n");
      printf("     MENU\n\n");
      printf("1.)  Adjust Left Motor\n");
      printf("2.)  Adjust Right Motor\n");
      printf("3.)  Turn On Interrupts.\n");
      printf("4.)  Turn Off Interrupts.\n");
      printf("5.)  Set Speed.\n");
      printf("6.)  Toggle DeBug.\n");
      printf("7.)  Toggle Menu Display.\n");
      printf("8.)  Toggle Solenoid 1.\n");
      printf("a.)  Select Speed/State.\n");
      printf("b.)  RTI Time Set.\n");
      printf("c.)  Turn On AD System.\n");
      printf("d.)  Read AD Ports.\n");
      printf("e.)  Toggle Sensor Check.\n");
      printf("f.)  Wireless Mode On.\n");
      printf("u.)  Toggle Gate");
      printf("9.)  Exit\n");
    }
  }

/*Process Input*/
```

```
void processSelection(int menuSelection)
{
  int a, b;
  if(debug)
    printf("Processing Selection:  %d\n", menuSelection);

  switch (menuSelection)
  {
    case '0':
    //Re-Sync
      break;
    case '1':
      if (wirelessMode == 0)
      {
        pulseWidth_TOC2 = parseInputToInt();
        if (debug)
          printf("Servo 1 Updated: pulseWidth = %d\n", pulseWidth_TOC2);
      }
      break;
    case '2':
      if (wirelessMode == 0)
      {
        pulseWidth_TOC3 = parseInputToInt();
        if (debug)
          printf("Servo 2 Updated:  pulseWidth = %d\n", pulseWidth_TOC3);
      }
      break;
    case '3':
      if (wirelessMode == 0)
      {
        asm("cli");
      }
      break;
    case '4':
      if (wirelessMode == 0)
      {
        asm("sei");
      }
      break;
    case '5':
      if (wirelessMode == 0)
      {
        if (debug)
          printf("Enter Row# You Want Modified.\n");
        a = getchar();
        a = a - 0x30;
        if (debug)
          printf("Enter Column# You Want Modified.\n");
        b=getchar();
        b = b - 0x30;


        menuSelection = parseInputToInt();
        speed[a][b][0] = menuSelection;

        menuSelection = parseInputToInt();
        speed[a][b][1] = menuSelection;

        if (debug)
          printf("(%d, %d) Updated: %d - %d", a, b, speed[a][b][0], speed[a][b][1]);
      }
      break;
    case '6':
      if (wirelessMode == 0)
      {
        debug = (debug + 1)%2;
      }
      break;
    case '7':
      if (wirelessMode == 0)
```

```
    {
      menuDisplay = (menuDisplay + 1)%2;
    }
    break;
  case '8':
    if (wirelessMode == 0)
    {
      if (shooting)
      {
        CLEAR_BIT(PORTA, 0x10);
        shooting = 0;
      }
      else
      {
        SET_BIT(PORTA, 0x10);
        shooting = 1;
      }

      if (debug)
        if (shooting)
          printf("Shooting Solenoid on PORTA - 0x10.\n");
        else
          printf("Shooting stopped on PORTA - 0x10.\n");
    }
    break;
  case 'a':
    if (wirelessMode == 0)
    {
      if (debug)
        printf("Enter Speed You Want:\n");
      a = getchar();
      a = a - 0x30;
      b = getchar();
      b = b-0x30;
      setSpeed(a, b);
    }
    break;

  case 'b':
    if (wirelessMode == 0)
    {
      if (debug)
        printf("What is the new RTII value you want.\n");
      rtiTime = parseInputToInt();

      if (debug)
        printf("RTI Time:  %d\n", rtiTime);
    }
    break;
  //Quick Speeds and Commands
  case 'c':
    if (wirelessMode == 0)
    {
      if (ADStatus)
      {
        CLEAR_BIT(OPTION, 0x80);
        ADStatus = 0;
      }
      else
      {
        SET_BIT(OPTION, 0x80);
        ADStatus = 1;
      }

      if (debug)
        if (ADStatus)
          printf("AD System On.\n");
        else
          printf("AD System Off.\n");
    }
```

```
      break;
    case 'd':
      if (wirelessMode == 0)
      {
        readAD();
      }
      break;
    case 'e':
      if (wirelessMode == 0)
      {
        if (sensorCheckEnable)
        {
          sensorCheckEnable = 0;
          printf("Sensor Check:  Disabled.\n");
        }
        else
        {
          sensorCheckEnable = 1;
          printf("Sensor Check:  Enable.\n");
        }
      }
      break;
    case 'f':
      if (wirelessMode == 0)
        wirelessMode = 1;
      break;
    case 'u':
    //Toggle Gate
      if (pulseWidth_TOC5 == gateUp)
      {
        pulseWidth_TOC5 = gateDown;
        if (debug)
          printf("Gate is Down\n");
      }
      else
      {
        pulseWidth_TOC5 = gateUp;
        if (debug)
          printf("Gate is Up\n");
      }

      break;
    case 'v':
    //Shoot Solenoid
      shootSolenoid();
      break;
    case 'w':
    //Right
      setSpeed(2,3);
      break;
    case 'x':
    //Left
      setSpeed(2,1);
      break;
    case 'y':
    //Forward
      setSpeed(1,2);
      break;
    case 'z':
    //Stop
      setSpeed(2,2);
      break;
    default:
      printf("What the fuck did you say?\n");
      break;
  }
  if (sensorCheckEnable)
    checkSensors();
  printf("ready\n");
}
```

```
int readAD(void)
{
  if (!ADStatus)
  {
    printf("AD System is not turned on.");
    return 0;
  }
  ADCTL = 0x14;
  while ( (ADCTL & 0x80) == 0 );
  if (debug)
    printf("AD Complete:\n\n\tChannel 4:  %d\n\tChannel 5:  %d\n\tChannel 6: %d\n\tChannel 7: %d\n\n", ADR1, ADR2, ADR3,
ADR4);
  return 1;
}

void checkSensors(void)
{
  if (readAD())
  {
  //Check Left Bumper
    if (leftBumper  > LEFT_BUMPER_THRESHOLD)
    {
    //Reverse 1x & Turn Right
      setSpeed(4,2);
      while (rtiInterrupts < rtiTime);
      setSpeed(4,2);
      while (rtiInterrupts < rtiTime);
      setSpeed(4,2);
      while (rtiInterrupts < rtiTime);
      setSpeed(2,4);
      while (rtiInterrupts < rtiTime);
      setSpeed(2,4);
      while (rtiInterrupts < rtiTime);
    }
  //Check Right Bumper
    if (rightBumper > RIGHT_BUMPER_THRESHOLD)
    {
    //Reverse 1x & Turn Left
      setSpeed(4,2);
      while (rtiInterrupts < rtiTime);
      setSpeed(4,2);
      while (rtiInterrupts < rtiTime);
      setSpeed(4,2);
      while (rtiInterrupts < rtiTime);
      setSpeed(2,0);
      while (rtiInterrupts < rtiTime);
      setSpeed(2,0);
      while (rtiInterrupts < rtiTime);
    }
  }
}
void shootSolenoid(void)
{
  pulseWidth_TOC4 = shootWidth;
  shooting = 1;
  rtiInterrupts = 0;
  SET_BIT(TMSK2, 0x40);
}

int parseInputToInt(void)
{
  int input = 0;
  int n = 0;

  if (debug)
    printf("Getting Int Input....\n");

  input = ((int)getchar() - 0x30)*10000;
  input = input + ((int)getchar() - 0x30)*1000;
```

```
    input = input + ((int)getchar() - 0x30)*100;
    input = input + ((int)getchar() - 0x30)*10;
    input = input + ((int)getchar() - 0x30);

    return input;
  }

  void setSpeed(int a, int b)
  {
    pulseWidth_TOC2 = speed[0][5][0] + speed[a][b][0];
    pulseWidth_TOC3 = speed[0][5][1] + speed[a][b][1];
    rtiInterrupts = 0;
    SET_BIT(TMSK2, 0x40);
  }

/* Interrupt Routines */
  void isrTOC2(void)
  {
    int temp = period - pulseWidth_TOC2;

    if (pulseOn_TOC2)
    {
      TOC2 = TOC2 + temp;
      SET_BIT(TCTL1, 0xC0);
      pulseOn_TOC2 = 0;
    }
    else
    {
      TOC2 = TOC2 + pulseWidth_TOC2;
      SET_BIT(TCTL1, 0x80);
      CLEAR_BIT(TCTL1, 0x40);
      pulseOn_TOC2 = 1;
    }

    CLEAR_FLAG(TFLG1,0x40);
  }

  void isrTOC3(void)
  {
    int temp = period - pulseWidth_TOC3;

    if (pulseOn_TOC3)
    {
      TOC3 = TOC3 + temp;
      SET_BIT(TCTL1, 0x30);
      pulseOn_TOC3 = 0;
    }
    else
    {
      TOC3 = TOC3 + pulseWidth_TOC3;
      SET_BIT(TCTL1, 0x20);
      CLEAR_BIT(TCTL1, 0x10);
      pulseOn_TOC3 = 1;
    }
    CLEAR_FLAG(TFLG1,0x20);
  }

  void isrTOC5(void)
  {
    int temp = period - pulseWidth_TOC5;

    if (pulseOn_TOC5)
    {
      TOC5 = TOC5 + temp;
      SET_BIT(TCTL1, 0x03);
      pulseOn_TOC5 = 0;
    }
    else
    {
      TOC5 = TOC5 + pulseWidth_TOC5;
```

```
    SET_BIT(TCTL1, 0x02);
    CLEAR_BIT(TCTL1, 0x01);
    pulseOn_TOC5 = 1;
  }

  CLEAR_FLAG(TFLG1,0x08);
}

void isrTOC4(void)
{
  int temp = period - pulseWidth_TOC4;

  if (pulseOn_TOC4)
  {
    TOC4 = TOC4 + temp;
    SET_BIT(TCTL1, 0x0C);
    pulseOn_TOC4 = 0;
  }
  else
  {
    TOC4 = TOC4 + pulseWidth_TOC4;
    SET_BIT(TCTL1, 0x08);
    CLEAR_BIT(TCTL1, 0x04);
    pulseOn_TOC4 = 1;
  }

  CLEAR_FLAG(TFLG1,0x10);
}


void rtii(void)
{
//Reset To Idle Position After aspecified Time
  if (rtiInterrupts++ > rtiTime)
  {
    pulseWidth_TOC2 = speed[0][5][0];
    pulseWidth_TOC3 = speed[0][5][1];
    CLEAR_BIT(TMSK2, 0X40);
  //Stop Shooting Solenoid If Shooting
    if (shooting == 1)
    {
      pulseWidth_TOC4 = shootWidth;
      rtiInterrupts = 0;
      SET_BIT(TMSK2, 0x40);
      shooting = 2;
    }
    else if (shooting == 2)
    {
      pulseWidth_TOC4 = stopShootWidth;
    }
  }
  CLEAR_FLAG(TFLG2,0x40);
}

void initializeServos()
{
  printf("Initializing Servos:  Begin.\n");
  *((void (**)())0xffe6)  = isrTOC2;    /*  Set Interrupt Vector:  TOC2 */
  *((void (**)())0xffe4)  = isrTOC3;    /*  Set Interrupt Vector:  TOC2 */
  *((void (**)())0xffe2)  = isrTOC4;    /*  Set Interrupt Vector:  TOC2 */
  *((void (**)())0xffe0)  = isrTOC5;    /*  Set Interrupt Vector:  TOC5 */
 *((void (**)())0xfff0)  = rtii;   /*  Set Interrupt Vector:  RTII */

  CLEAR_BIT(OC1M, 0xff);/*  Disable OC1 Access */
  SET_BIT(TMSK1, 0x40);             /*  Enable TOC2 Interrupt */
  SET_BIT(TMSK1, 0x20);             /*  Enable TOC3 Interrupt */
  SET_BIT(TMSK1, 0x08);             /*  Enable TOC3 Interrupt */
  SET_BIT(TMSK1, 0x10);             /*  Enable TOC4 Interrupt */
  CLEAR_FLAG(TFLG1, 0x40);  /*  Clear Possible Flag:  TOC2 */
  CLEAR_FLAG(TFLG1, 0x20);        /*  Clear Possible Flag:  TOC3 */
```

```
  CLEAR_FLAG(TFLG1, 0x08);          /*  Clear Possible Flag:  TOC3 */
  CLEAR_FLAG(TFLG1, 0x10);          /*  Clear Possible Flag:  TOC4 */
  printf("Initializing Servos:  Complete\n");
}
```