Pele II
Soccer Playing Robot

Karina J. DaCosta
College of Electrical Engineering
University of Florida
December 4, 2001
EEL 5666

Table of Contents

Abstract

Various forms of soccer-style games have been around since Roman times. The United States as a matter of fact was the first British colony to start playing soccer-style games. Some form of football was played in the Colonies as far back as the establishment of the original Jamestown settlement in 1609. Imaging equipping a robot with the right components to play American football, it would be a brilliant idea. Even more brilliant would be equipping a robot to play a game known through out the world. A game that has an old history and that is still popular today in other forms such as football, ballown, etc. Thus, creating a soccer playing robot is not only a challenge but part of history.

Executive Summary

Pele II, was able to find the goal and score. It randomly went around its enclosed environment searching for the soccer ball. It also avoided obstacles while it searched for the soccer ball. It uses a breakbeam to detect the ball and two IRs for collision avoidance. When the ball breaks the beam the microprocessor detects a different signal indicating the ball is there. Once Pele II successfully detects the ball it starts looking for the goal. It slowly turns until it detects a signal from the goal (beacon). There are two collimate 40 kHz IR cans that are used to detect the signal from the goal. Once both IRs find this signal, the goal has been found. Once it finds the goal it starts accelerating towards the goal and finally shoots the ball out into the

goal. The mechanism it uses to shoot the ball out is a solenoid. When the ball goes inside the goal it touches a flex sensor. This flex sensor is connected to an RF transmitter which sends a signal to the RF receiver on the robot. When the robot receives the signal it turns LEDs on and off indicating it has scored.

Introduction

Pele II is an autonomous soccer playing robot. The main objective of the robot is to score a goal. Pele II simply looks for a ball and aligns itself with the goal and shoots for the goal.

Integrated System

Pele II is operated by using the MRC11 and MRSX01 from Mekatronix. Components on these boards were soldered using the Talrik's manual. Pele II contains two partially hacked servos, three 40 kHz IR cans -- one for the breakbeam and two for beacon detection, and two IR sensors for collision avoidance. The beacon is a 42 kHz signal that is generated by a 555 timer chip. There is one solenoid. This solenoid is not directly connected to the board for the microprocessor's safety. It's connected to a photo oscillator, to oscillate noise. In the goal there is a flex sensor connected to an op-amp which connects to an RF transmitter.

Mobile Platform

I used a plain platform. A flat board with an angle slot (see Figure 1a). The design made for the Talrik's switches were used for Pele II. The height of the platform was adjusted so that the solenoid could be placed underneath

the platform. The solenoid's thickness and the height of the wheels were taken into consideration when the AutoCAD design was made.


Figure 1a

Actuation

Pele II rides on two 3-inch Du-Bro wheels and a rear carriage bolt (instead of a caster). The Du-Bro wheels were purchased from Mekatronix. Each of the two wheels has a partial servo hack, since each servo needs to behave similarly to a drive motor. By having a partial servo hack, it is able to control the output shaft's velocity. In addition, the speed is controlled via Pulse Width Modulated Signal (PWM). As far as stability is concerned, placing a rear caster made Pele II gear of a little when it was in slow-motion trying to go straight. The rear carriage bolt on the other hand , created more stability, since it does not move in any direction. The only shortcoming with the carriage bolt is it renders movement on rug surfaces difficult.

Sensors

Mekatronix IR SENSORS

The Infrared Sensors (IR) that were bought from Mekatronix did not need hacking. These IRs were used for collision avoidance because they have great range. They are able to detect better than the regular Sharp Cans. They can also be combined with the Sharp Cans and not have to worry about signal interference with one another

since they operate at different frequencies.

SHARP CAN SENSORS

The 40 kHz IRs did however need to be hacked. This is because they are digital devices that will only produce digital outputs. These IRs have to be modified to make them analog devices. This modification will allow the robot to know how far the objects are, or, if they are being used as a breakbeam, they'll detect an interruption. Pele II uses three IRs -- one as a breakbeam combined with an emitter and two to detect the beacon. This beacon emits a 42 kHz modulated signal which does not interfere with the Mekatronix IRs used for collision avoidance.

BUMP SENSOR

The bumper sensor is essentially a switch. This switch closes when it is pressed and the signal is received by the microprocessor PE0. In my design when the bump switch is pressed, Pele II stops moving.

Special Sensor

Besides having IRs as sensors, Pele II is also equipped with a bend sensor to detect a goal score. The Bend sensor is a unique component that changes resistance when bent. An unflexed sensor has a nominal resistance of 10,000 ohms (10 K). As the flex sensor is bent, the resistance gradually increases. When the sensor is bent at 90 degrees its resistance will range between 30-40 K ohms. For my design, I didn't have to worry about trying to bend the flex sensor 90 degrees since the flex sensor has a thickness of .019 inches is highly sensitive to a simple tap which is all Pele II needs. As mentioned before, the flex sensor doesn't need to bend 90 degrees; as long as the ball taps the flex sensor it will create a noticeable resistance. Here *noticeable* is defined as any value greater than the nominal value. If R_2 = 33 k ohms, and the

input voltage is 5 volts, then the output voltage will be in the range from 2.8 to 4 volts. The positive power to the op-amp can be 5 volts and the negative power to the op-amp can be 0 volts. A better value is obtained if R_2 = 33 k ohms is changed to R_2 = 22 k ohms (see Figure 2).

Figure 1

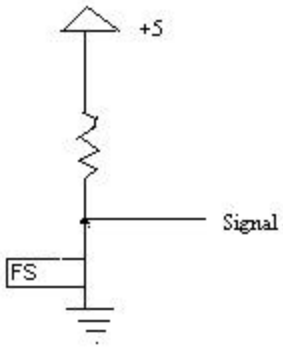


Figure 2

The reason for claiming this as a special sensor is because it will not stand

on its own; it will be used in conjunction with another device to exploit its unique feature. For this design, the optimal place where the flex sensor can be placed is at the goal. It keeps track of the goal. Once the ball taps the flex sensor, the flex sensor along with the 22k (voltage divider) will send a signal to an op-amp. The op-amp will send out a signal to a transmitter which will later be received by a receiver that will be mounted on the robot. Let me regress and elaborate

more on how the op-amp works in my design. It has two inputs, the value obtained from the voltage divider goes into +V op-amp input. I have placed a potentiometer going into –V op-amp input. This potentiometer allows me to match the voltage to that of the voltage divider during its initial stage – the initial stage meaning values of the 22k ohms and nominal value of the flex sensor, 10.3k. By performing a voltage divider, I obtained 3.4 volts coming out of the voltage divider during its relaxed stage. Therefore, 3.4 volts will go into +V input and the potentiometer must be matched to 3.4 volts in the other input in order to not produce an output during its relaxed stage and to produced an output during a tap stage, when the ball hits the flex sensor (See Figure 3).

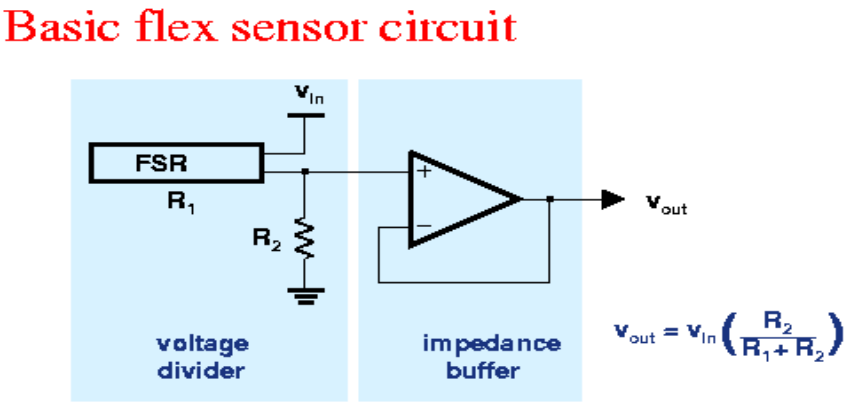


Figure 3

During its tap stage, the op-amp with go high sending a high signal to the transmitter. The wireless transmitter transmits the signal to the receiver via an λ/4 antenna. The signal is picked up by another λ/4 antenna that will be connected to the receiver which will be mounted on the robot. Please

refer to Figure 4 for better understanding. In sum, I have combined a flex sensor with a receiver and a transmitter, making all three components account for a special sensor.

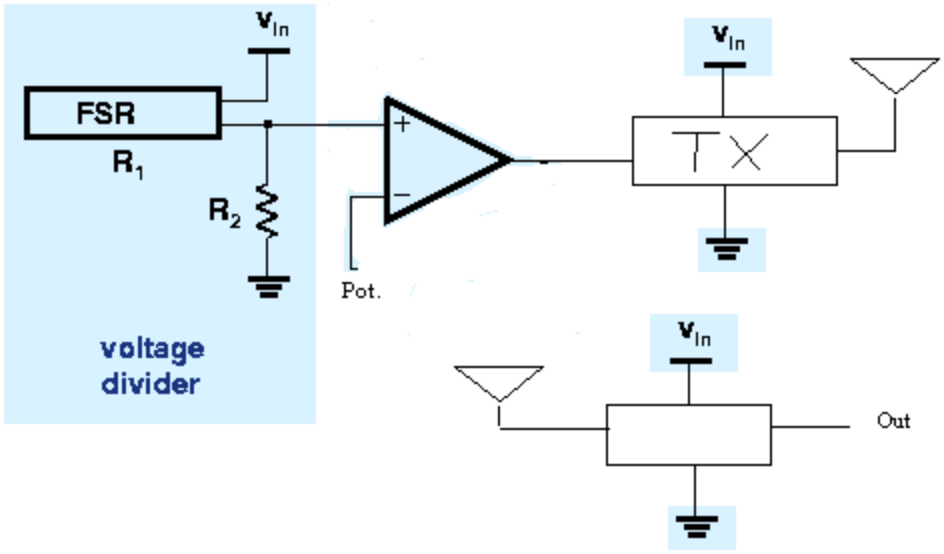
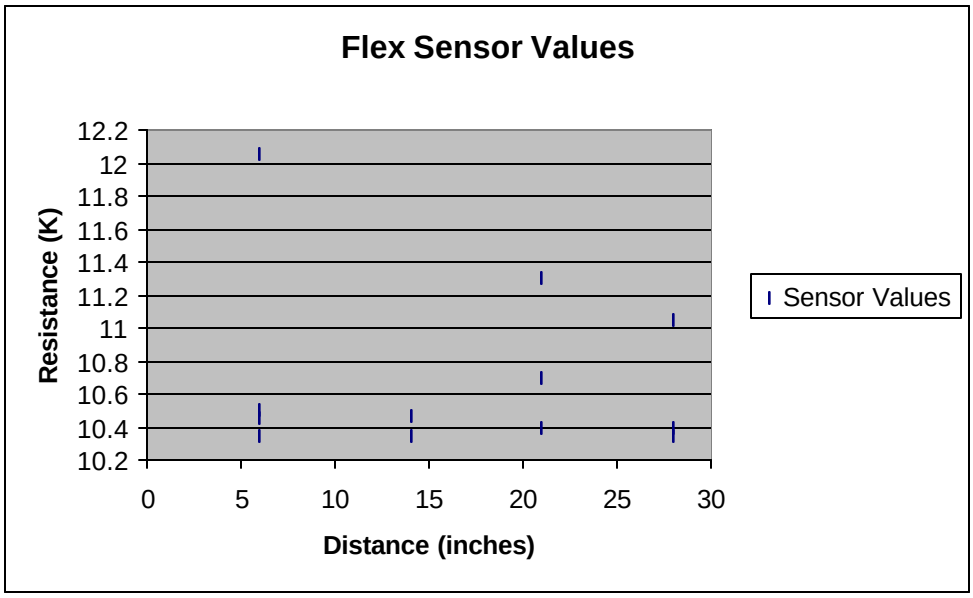


Figure 4

The values that I obtained when I tested the special sensor did not fit a straight line approximation because there were too many factors to consider: low batteries, all not centered, which area of the flex sensor was tapped by the ball, etcetera (See chart 1). (Chart 1)



Behaviors

The robot has four behaviors: 1) Collision avoidance 2) Identifying the ball 3) aligning the ball with the goal 4) Making the goal.

Lessons Learned

When I made the photo oscillator circuit I burnt the chip because I left out a pull-up resistor (470 ohms or 1k ohms). I also burned a few LEDs when I was testing the circuit I had built for the 40 kHz modulated signal located at the goal. I forgot to place a pull up resistor in series with the LEDs. So now when ever I built a circuit that either I obtained from a TA or a schematic from the Web I make sure I'm not missing a pull up resistor, if needed. Also it's a good habit to test circuits you built with a multimeter or logic probe and if available an oscilloscope. I had difficulties with the servos obtained from Mekatronix. With respect to hacking, they were more difficult to hack than servos from previous semesters. In the end, once I calibrated the servos I found myself having to superglue the potentiometer at the 50 % duty cycle in order to prevent it from losing the 50% duty cycle calibration. Also keep in mind that it is **almost impossible for servos to go in a straight line**. Try **not** to supply more than 6 Volts to the servo power unless

you want to test how fast they burn (this only applies to Mekatronix servos since they're what I used) -- see Figure 1. Also sometimes schematics on the Web are not correct, so make sure to either get a second opinion or an extra part in case the first one burns.

Software hints

I learned that usually timing is everything as far as trying to make the robot turn or even go in a fairly straight line. So you should give yourself plenty of time to test the hardware with the software.

Conclusion

The project was successful. The behaviors I proposed were realistic which contribute to the success of the project. Don't give up if things start burning, just learn from your or other people's mistakes. Also constant hard work and no procrastination pays off.

Appendix A.

```
/***********Karina J. DaCosta*************/
/****************PeleII*********************/
/***********Avoids Obstacles***************/

#include <stdio.h>
#include <tkbase.h>
#define THRESHOLD 128
#define RIGHT_IR analog (2)
#define LEFT_IR analog(3)
#define L 0  // left servo
#define R 1 //right servo
#define SPEED_1 1000
#define SPEED_2 5000
#define BUMP_STOP 79
#define BEAM_THRESHOLD 129
#define BREAKBEAM analog(4)
#define BEACONR      analog(5)//robots'right
#define BEACONL      analog(7)
#define BEACON_THRESHOLD 97
#define ANTENNA analog(6)

/*IR emitter output port driver, the output latch at address 0xffb9 */
#define IRE_OUT   *(unsigned char *)(0xffb9)

/*Constant for driving all the 40KHz modulated IR emitters on when loaded into IRE_OUT */
#define DIGIT_ON 0x3d
#define SOL_ON 0x3f

/*Constant for turning pin 2 40KHz modulated IR emitters off when loaded into IRE_OUT */
#define DIGIT_OFF 0x3d

#define GOAL_OFF 0x3d
#define GOAL_ON  0xfd

int state=0;
int val,l_ir, r_ir,speedl, speedr ;
int intPrevValue = 999;
int intCurrentValue = 999;
int intDifference = 0;

void read_sensors();
void init();
void print_sensor_values();


void main()
{
  init();
  servo(L, 0);
```

```
  servo(R, 0);
  while(1)
  {
          print_sensor_values();
  }
}

/*** END MAIN ***/

void init()
{
  init_analog();
  init_servos();
  init_clocktk();
  IRE_OUT=DIGIT_OFF;
  IRE_OUT=DIGIT_ON;

  //wait(50);

}

void print_sensor_values()
{
    unsigned int  rb;
          rb = rear_bumper();       /*Returns rear bumper value*/
    printf("\n \n analog 2: {%d}\n", RIGHT_IR);
    printf("analog 3: {%d}\n", LEFT_IR);
    r_ir = RIGHT_IR;//new from here
    l_ir = LEFT_IR;

   if(state==0){
     if(l_ir >THRESHOLD || r_ir >THRESHOLD ){
       if (l_ir > THRESHOLD){
                          servo(R, 2950);//slow moving
                servo(L, 3041);
                          wait(100);//wait a bit
                          servo(R, 3141);//move back
                          servo(L, 2950);
                          wait(300);
                          servo(R, 4000);//turn right
                          servo(L, 2950);
                          wait(400);
                          servo(R, 2000);
                          servo(L, 4000);

            }

            else if (r_ir >THRESHOLD) {
               servo(R, 2950);//slow moving
                servo(L, 3041);
                          wait(100);//wait a bit
```

```
                        servo(R, 3141);//move back
                        servo(L, 2950);
                        wait(300);
                        servo(R, 2950);//turn left
                        servo(L, 4000);
                        wait(400);
                        servo(R, 2000);
                        servo(L, 4000);
            }

            }//end if l_ir or r_ir greater than threshold
            else {
                    servo(R, 2930);//slow moving
                servo(L, 3041);
            }
            if ( rb == BUMP_STOP ){
                    state=5 ;
            }
            if (BREAKBEAM < BEAM_THRESHOLD){
                    state = 2 ;
            }
           }//end state 0

/*********************BALL DETECTION*****************************/

           else if ( state == 2 ){

                    if ( BREAKBEAM < BEAM_THRESHOLD ){
                            printf("Ball Breakbeam 4: {%d}\n", BREAKBEAM);
                            servo(R, 2930);//slow moving
                servo(L, 3041);
                wait(50);
                servo(R, 0);//stop moving
                servo(L, 0);

                            state = 3;
                    }
                    else state = 0;
                    if( rb == BUMP_STOP ) state = 5 ; // end rb

           }// end state 2

/************** GOAL DETECTION **********************/

            else if( state == 3 ){
           // printf("goal detection");
                    printf("right beacon5: {%d}\n", BEACONR);
           printf("left beac analog 7: {%d}\n", BEACONL);

                    if ( BEACONR > BEACON_THRESHOLD && BEACONL >
BEACON_THRESHOLD ){
```

```
                printf("Found Goal");
                state = 4 ;

            }// end beacon > BEACON_THRESHOLD

            else if ( BEACONR < BEACON_THRESHOLD && BEACONL >
BEACON_THRESHOLD ){
                printf("LEFT FOUND - TURNING RIGHT-------");
                servo(R, 2930);
                servo(L, 3041);
                wait(200);

                servo(R, 0);//turn robot's right
                servo(L, 3041);
                wait(300);

                servo(R, 0);
                servo(L, 0);
                if (BEACONR > BEACON_THRESHOLD && BEACONL >
BEACON_THRESHOLD){
                    printf("Found Goal");
                    state = 4 ;
                }// end beacon > BEACON_THRESHOLD


            }// end else if
            else if ( BEACONR > BEACON_THRESHOLD && BEACONL <
BEACON_THRESHOLD ) {
                printf("RIGHT FOUND - TURNING LEFT********");
                servo(R, 2930);// slow moving
                servo(L, 3041);
                wait(200);
                servo(R, 2930);//turn left
                servo(L, 0);
                wait(100);
                servo(R, 0);
                servo(L, 0);
                //check if goal found
                if (BEACONR > BEACON_THRESHOLD && BEACONL >
BEACON_THRESHOLD){
                    printf("Found Goal");
                    state = 4 ;
                }// end beacon > BEACON_THRESHOLD

             }// end else if beacon
             else if (BEACONR < BEACON_THRESHOLD && BEACONL <
BEACON_THRESHOLD){
                    servo(R, 2930);//turn right
                    servo(L, 0);
                    wait(200);
                    servo(R, 0);
```

```
                                servo(L, 0);
                                if (BEACONR > BEACON_THRESHOLD && BEACONL >
BEACON_THRESHOLD){
                                        printf("Found Goal");
                                        state = 4 ;
                                }

                 }// end else if they are both less than 97

                if ( rb == BUMP_STOP ){
                        state = 5 ;
                }
        }// end state 3
/******************KICK BALL******************/


                else if( state == 4 ){

                        if ( BREAKBEAM < BEAM_THRESHOLD ){
                           servo(R, 2930);//slow moving
                servo(L, 3041);
                                wait (200);
                                servo(R, 1000);
                                servo(L, 5000);
                                wait (400);
                                IRE_OUT = SOL_ON;
                                wait (400);
                                IRE_OUT = DIGIT_OFF;
                                state = 6;

                        }
                        else if( BREAKBEAM >= BEAM_THRESHOLD ) {
                                IRE_OUT=DIGIT_OFF;
                        wait(400);
                        servo(R, 2930);//slow moving
                servo(L, 3041);
                                wait (200);
                                state = 4 ;
                        }
                        if ( rb == BUMP_STOP ){
                                state = 5 ;
                        }

                }// end state 4

        else if ( state == 5){
          if ( rb == BUMP_STOP ){
                        servo(R, 0);// stop the servos
                        servo(L, 0);
                        printf("\n \n analog 0: {%d}\n", rb);
                wait(100);
```

```
            }
          }// end state 5 which is bumper to stop servos



/**********  STOP EXECUTION  ***************/
          else if ( state == 5 ){
              if ( rb == BUMP_STOP ){
                        servo(R, 0);// stop the servos
                        servo(L, 0);
            printf("\n \n analog 0: {%d}\n", rb);
      wait(100);
            }

          }// end state 5

/**********  SENSE GOAL  *********************/

          else if ( state == 6 ){
                    intPrevValue = ANTENNA;
                    servo(L, 0);
                    servo(R, 0);
          printf("\n \n previous value: {%d}\n", intPrevValue);
          if( intPrevValue != 999 && intCurrentValue != 999 ){
                    intDifference = abs(intPrevValue - intCurrentValue);
                    if ( intDifference >= 165 ){
                              printf("\n******* GOL!!! *******\n");
                              state = 7;
                    }
          }
          intCurrentValue = ANTENNA;
          printf("\n \n current value: {%d}\n", intCurrentValue);
          if( intPrevValue != 999 && intCurrentValue != 999 ){
                    intDifference = abs(intPrevValue - intCurrentValue);
                    printf( "%d", intDifference );
                    if ( intDifference >= 165 ){
                              printf("\n******* GOL!!! *******\n");
                              state = 7;
                    }
          }
                    if ( rb == BUMP_STOP ){
                              state=5 ;
                    }// end rb
          } // end state 6

/*********  FLASH "GOAL" SIGN  ***********/

    else if ( state == 7 ){
          int i = 0;
          for(i=0; i<5; i++){
                    IRE_OUT=GOAL_ON;
```

```
                    wait(500);
                    IRE_OUT=GOAL_OFF;
                    wait(500);
                }
                state = 5;
    } //end state 7


}
```