Steven Edouard

Final Report

Project: Parker-B

EEL5666 Spring 2011

Dr. Arroyo and Dr. Schwartz

Table of Contents

Executive Summary

Parker-B is an autonomous smart parallel parking robot. Parker-B uses sensors to detect parking spaces along a wall (which simulates cars parked on the side of a road). Parker-B then detects if an illegal parking space sign exist and if so, it will look for another parking space.

Parker-B begins its mission by wall following until it detects a parking space large enough. It turns its servo to the right and scans the parking space by turning the camera left and right until either the parking sign is either detected or not. If it is, it will continue to seek for another space; if it doesn't detect the sign it will negotiate itself into the parking space.

This autonomous system is composed of two main components, the Mobile Platform and the Vision Command Unit. The Mobile Platform is the physical robot you see in the demonstrations; the Vision Command Unit (VCU) works behind the scenes on the laptop providing vision services to the Mobile Platform. The VCU detects the parking sign, communicates its existence to the Mobile Platform and also controls the actuation of the camera mounted on the Mobile Platform.

1. 0 Abstract

The Parker-B project is a mobile autonomous robot utilizing computer vision algorithms, range finding techniques and a joint effort between two computer systems to negotiate itself into a parallel parking space. Parker-B will utilize an Atmel Xmega MCU to control the robot platform and use computer vision processing algorithms on a commercially available laptop and communicate via serial (using Bluetooth) to achieve its goal.

Parker-B will have a set of abilities to negotiate itself through an "urban" environment. The robot will have the ability to first identify a potential parking space using side-facing IR range finders and by identifying the parking space it will also measure the space of the parking space. The microprocessor then communicates to the laptop to obtain information on the legality of said parking space. The software on the laptop will then determine if the space is a "legal" parking space by checking the parking space for a sign that indicates that there is "No Parking". If the space is legal, the computer software will then instruct the Parker-B to attempt to negotiate its platform into the parking space.

The MCU then initiates a parking algorithm using IR range finders (mounted on the front , back and right side of the platform), which will negotiate the platform into the parking space.

2.0 Introduction

2.1 Background

In Florida, parallel parking seems to be a big problem with most Florida drivers. The idea of Parker-B is to be a prototype for an interesting demonstration of machine intelligence, parallel parking rules and techniques. When searching the Internet for self-parking robots, there are few robots that are able to identify a parking space, check some characteristics about that parking space and negotiate itself into that space. Most of these machines are simply programmed to negotiate the platform into some preset "parking space".

2.2 Scope and Objectives

The Parker-B robot will utilize side facing IR range finders, which will detect when a "car" is present on the "curb". Using this information Parker-B can detect and approximate the length of a parallel parking space. After this approximation Parker-B will detect for the presence of a "no parking" sign, indicating the legality of this parking space. Finally the system will negotiate itself into the parking space using IR range finders.

2.3 References to Literature

Leveraging OpenCV, this system can be trained to create a classifier (something which defines what the software should be looking for in an image), which will quickly detect "parking signs". Using Haar-like features training, OpenCV offers applications, which will allow developers to create these classifiers, which work well with well-defined objects. More on this method can be found at:

- **OpenCV haartraining tutorial:**
  http://note.sonots.com/SciSoftware/haartraining.html

  Using a Bluetooth serial connection the mobile platform will be able to

Although openly editable and not always accurate, Wikipedia defines Haar-like features very accurately.

- http://en.wikipedia.org/wiki/Haar-like_features

3.0 Integrated System

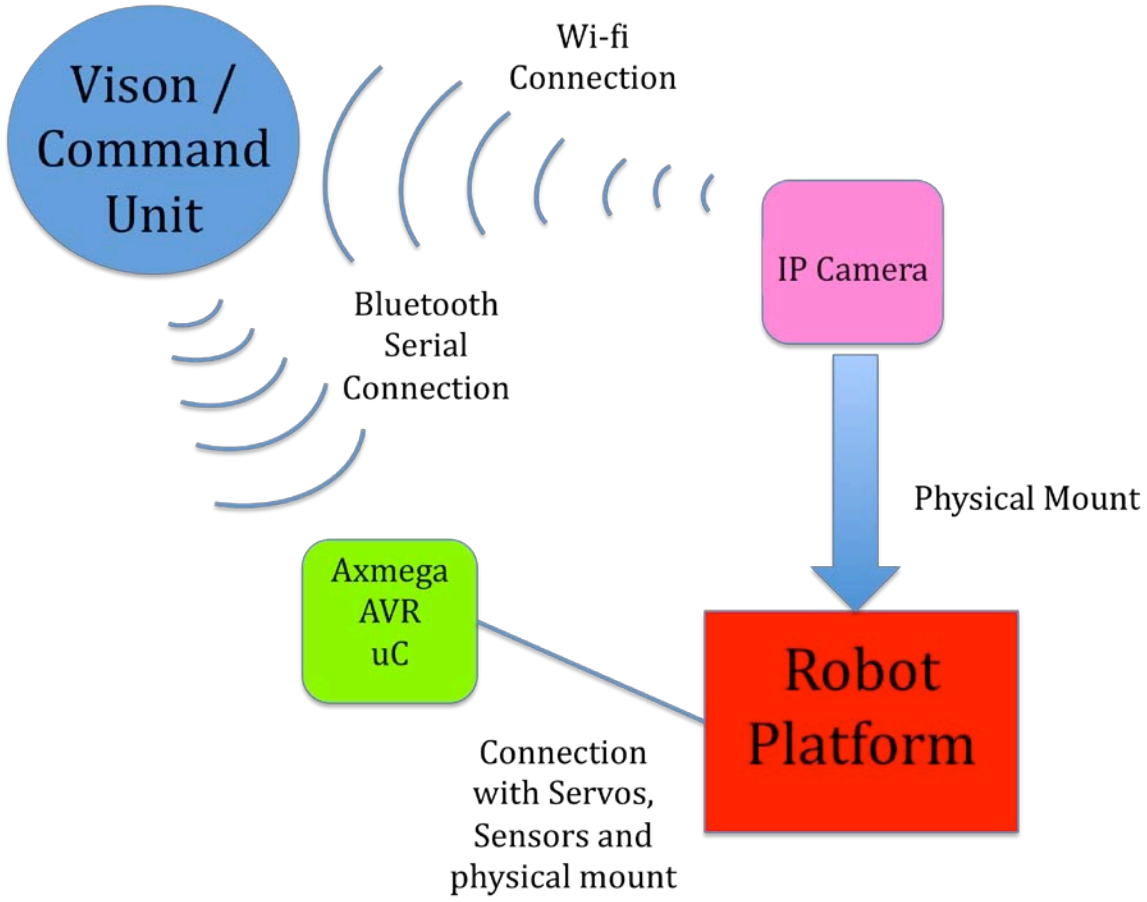

Figure 3.0 – Parker-B High-level integration block diagram

The diagram above shows a high level view of the integration of the Parker-B mobile autonomous parking system modules.

3.1 Vision / Command Unit (VCU)

As the name suggest, this module is responsible for commanding the robot platform and for handling all the computer vision. This module is a purely software component and will connect to the IP camera via HTTP protocol and obtain images from the camera. The software searches the image for what it recognizes as a "parking sign" and then communicates a series of commands to the Parker-B autonomous robot via Bluetooth serial connection.

The Vision / Command Unit leverages the OpenCV computer vision framework to perform vision detection. The "parking signs" used will be of the same

texture and well defined in space. This makes using OpenCV's Haar-like features training software (haartraining) able to detect these objects in the parking space possible by using a set of training images. Usually the set of images can be hundreds even to the thousands.

Once the Haar-training algorithm is complete a classifier cascade will be assembled which comes in the form of an XML file. This classifier cascade will (ideally) identify the object in question with reasonable accuracy and high speed. The object in question for this project is the "No Parking Sign".

3.2 ATXMega AVR uC

The ATXmega AVR microcontroller will also have some intelligence. Obstacle avoidance will be implemented in this module. The uC will also provide a command set for the **Vision / Command** unit to communicate to the autonomous platform if a parking space has been detected and will offer various motor movement commands such as:

- FORWARD
- STOP
- BACKWARDS
- TURN_LEFT
- TURN_RIGHT

The uC will also have the instructions for the parallel-parking algorithm. Although the above commands are available for use of the Vision Command Unit, the uC will be responsible for actuation to maneuver the platform in the parking space.

3.3 IP Camera

The IP camera is a commercial-of-the-shelf (COTS) module. Its function is to relay images back to the **Vision / Command Unit** by hosting a webserver on the camera itself using a Wi-fi connection.

3.4 Robot Platform

This module is purely hardware. It includes the body of the robot, the servos and the wheels they drive, IR sensors, LCD unit, IP Camera and battery packs. The platform will be designed in a CAD environment and put together and will allow for the mounting of the IP Camera , the ATXMega AVR uC board, and batteries.

3.5 Mission Sequence Diagram

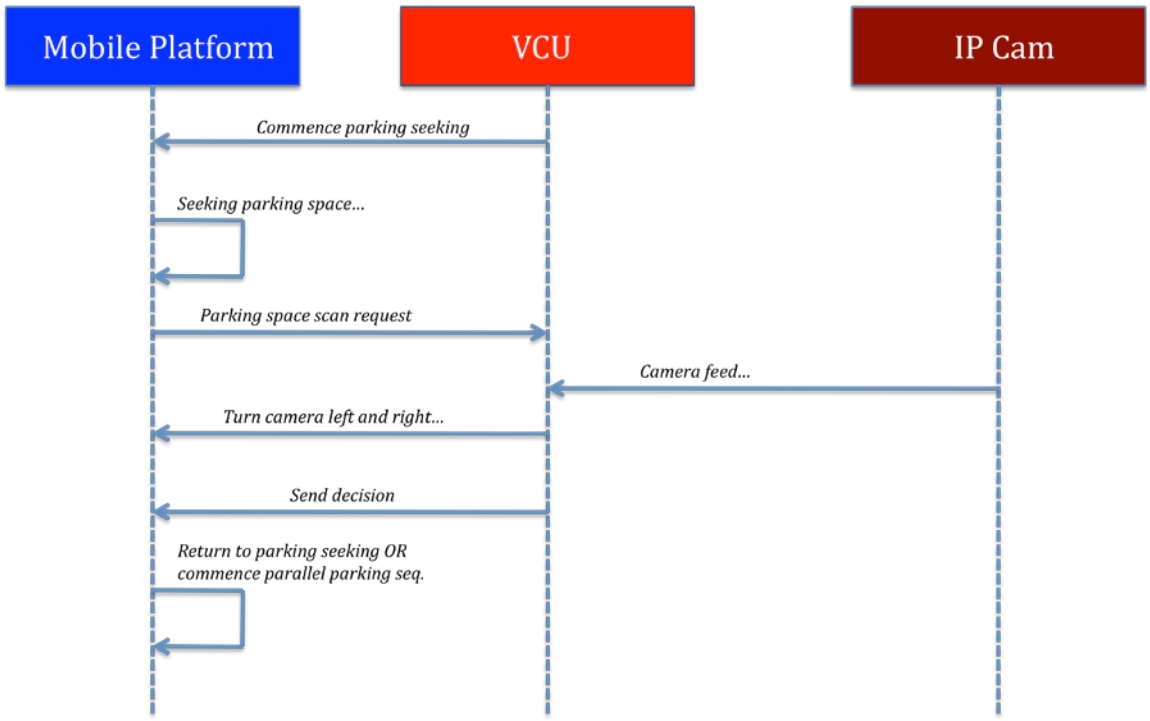

Figure 3.1 – System Integration Sequence Diagram

Figure 7.4 illustrates at a high-level the integration of the Parker-B system. The VCU initiates the Mobile Platform's behavior. A request is only sent when a parking space is detected; the camera feed is actually always being fed to the VCU, however only when the request from the Mobile Platform is sent the object detection of the "No Parking" sign takes a role in the decision process.

4.0 Mobile Platform

The mobile platform will be constructed of wood parts designed in SolidWorks CAD software and pieced together in a lab. The body will include mounts to micro controller board, the IP camera, 3 servos, 4 IR sensors and 4 bump sensors.

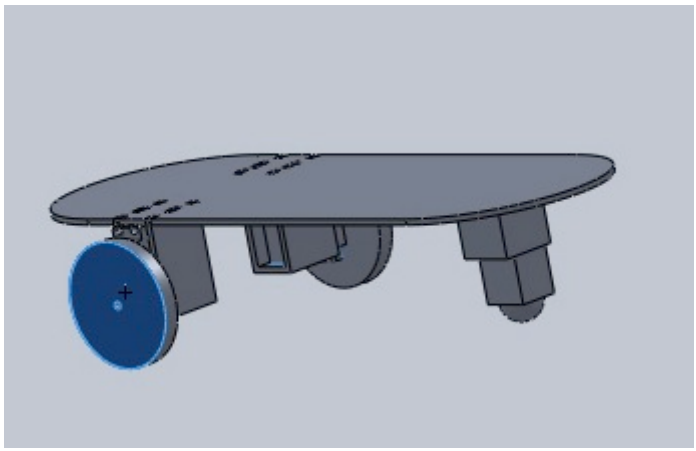

Figure 4.0 – Mobile Platform assembly CAD

The structure of the mobile platform will be a 3 wheeled design, two servo-driven wheels and one caster in the rear. The body will be an elongated rounded square (sort of resembling a skate board). The camera will sit upon a mount to provide the optimum view angle for the parking space. The platform will be simple but allow for plenty of space to mount all necessary sensors and equipment.

5.0 Actuation

Transit Actuation

The Parker-B system will have servos sitting below the Mobile Platform. Each of these servos drive a wheel which will allow for Parker-B to maneuver its way in all directions.

Parker-B doesn't need to move very quickly, nor does it need much power in actuation. The platform requires only enough torque to carry the batteries, board, camera and the platform itself. Servos can easily satisfy this without the need to have some separate driver device. The rational behind choosing servos for actuation is to keep the hardware simple and cost low.

The actuation driving the wheels is not always consistent with what is provided via PWM channels. Sometimes the actuation is randomly different and that is attributed mainly to low cost actuators. Software correction is implemented to mitigate risk in inaccurate actuation.

Camera Actuation

Parker-B mobile platform will have the IP camera mounted to a servo connected to the Parker-B Mobile Platform microprocessor however is controlled by both the Mobile Platform as well as the VCU. More information on this can be found in the System Integration section.

//TODO: REMOVE AND MOVE TO SYSTEM INTEGRATION

Upon parking space detection by the Mobile Platform the platform moves the servo 45 degrees to the right with respect to the direction of motion of Parker-B during parking seeking. The VCU then commands the servo left and right with approximately 2 degrees of motion per command.

6.0 Sensors

The sensors for Parker-B should be able to negotiate the autonomous vehicle through a pre-made "urban" environment. These sensors will be primarily responsible for ensuring that Parker-B can measure parking space distances and to avoid colliding with any objects.

IR Range-Finders

IR range-finding sensors will be placed toward the front, rear, and right side of the platform. The front and rear sensors will not be placed Bump sensors will be placed on the front and back of the platform, for the purpose of detecting an unintended collision.

The particular model of the range-finder to be used is the Sharp GP2D120XJ00F. The range for this sensor is published as 4-40cm which is a range that applies to the application of Parker-B.

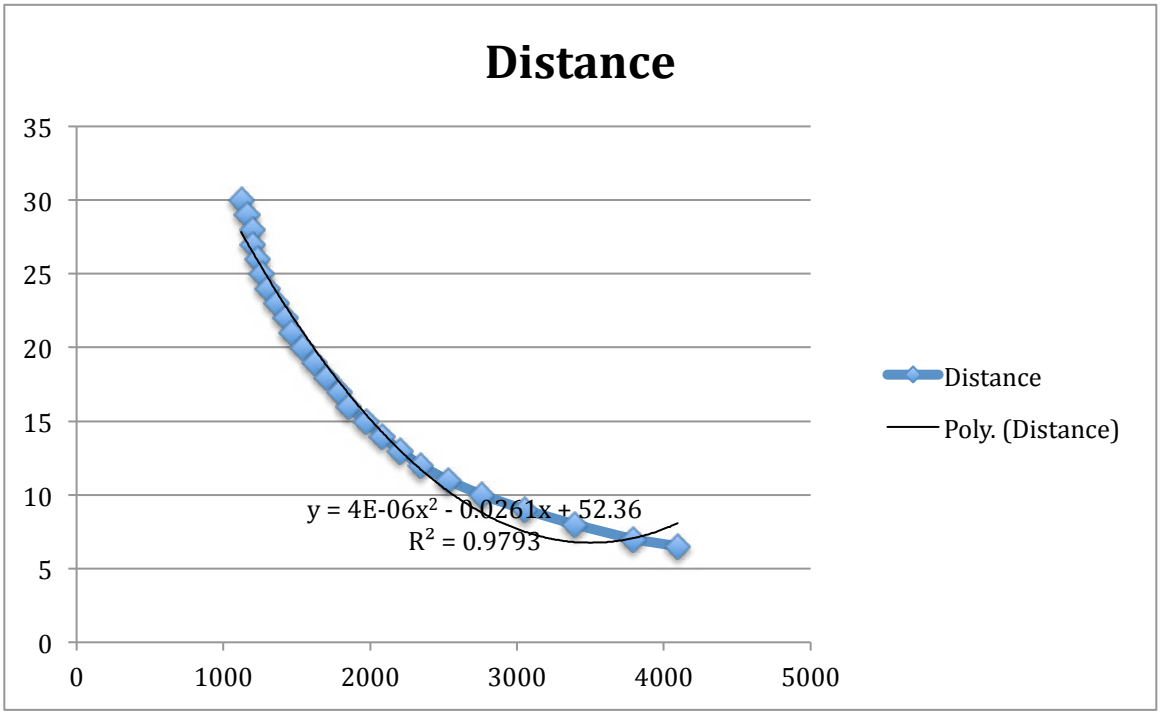

Figure 6.0 – A/D value (Y axis) vs. Distance (cm) (X axis)

IP Camera

The IP Camera's sole objective is to relay images of the parking space back to the Vision / Command Unit. It is from there the image is analyzed for characteristics of legal parking space.

Using OpenCV the Vision / Command Unit can search the video feed for the "no Parking Sign". As shown in the figure below.

Figure 6.1 – "No Parking Sign" object detection

Haar-like features training in OpenCv works by using rectangles of a predefined size to classify an area of an object. Face detection is a common application of Haar-like training. For example, in face detection, it is a common feature among images of faces that the area around the eyes is darker than the area around the cheeks. So using two rectangles that lie on the eye and below the eye could be used as a Haar-like feature to define a face.

In the application of Parker-B, this classifier uses OpenCv's Haar-like features training application to train the software to look for the parking sign based on thousands of positive and negative example images. A positive image is an example of an image with the object in question contained (in this case it is the "No Parking Sign"). Below is an example of one of the positive images used in the training set.

Figure 6.2 – “No Parking Sign”

Figure 6.1 – “No Parking Sign” object detection

OpenCv has tools to create positive image data sets by using one image of the object in question and thousands of negative images and places the object with different distortions and angles in the image. The software then uses the positive and negative images to define what is called a classifier cascade file which defines what OpenCv should be looking for in an image or video.

Parker-B’s object detection performance summary:

```
Total Number of Positive Images: 1000

Hits: 712

Misses: 288

False Alarms: 25
```

Figure 6.2 – Parker-b object detection performance summary

From the above output from OpenCv's classifier performance tool we can see that the hit rate is around 71% with a very small number of false alarms. This may sound not very well performing but from practical use, these statistics are sufficient enough for Parker-B to detect the "No Parking Sign" nearly all the time.

Bump Sensors

Bump sensors will simply be switches sensitive to bumps that indicates to the microcontroller that collision has occurred.

The software on the microcontroller should continuously poll this device or an interrupt can be assigned to these sensors.

Sensor Integration

The bump and IR sensors will be integrated with microcontroller. The IR sensors will be used as a first line of defense to collisions and to measure parking spaces. In the event of a collision, the bump switch will stop the algorithm the microcontroller is executing in order to stop or move the mobile platform away from the site of collision.

The IP Camera will be integrated with the Vision / Command Unit through an HTTP connection over a Wi-fi network and from there a Bluetooth interface between the Vision / Command Unit and the microcontroller modules will be used to seek potential parking spaces.

The VCU uses standard UNIX calls for writing to a serial device to transmit data via Bluetooth to the mobile platform. The OpenCV C++ API is used to interface the VCU to the HTTP connection to the IP camera on the mobile platform. Detailed code can be found in the Appendix.

7.0 Behaviors

Wall Following/Parking Seeking

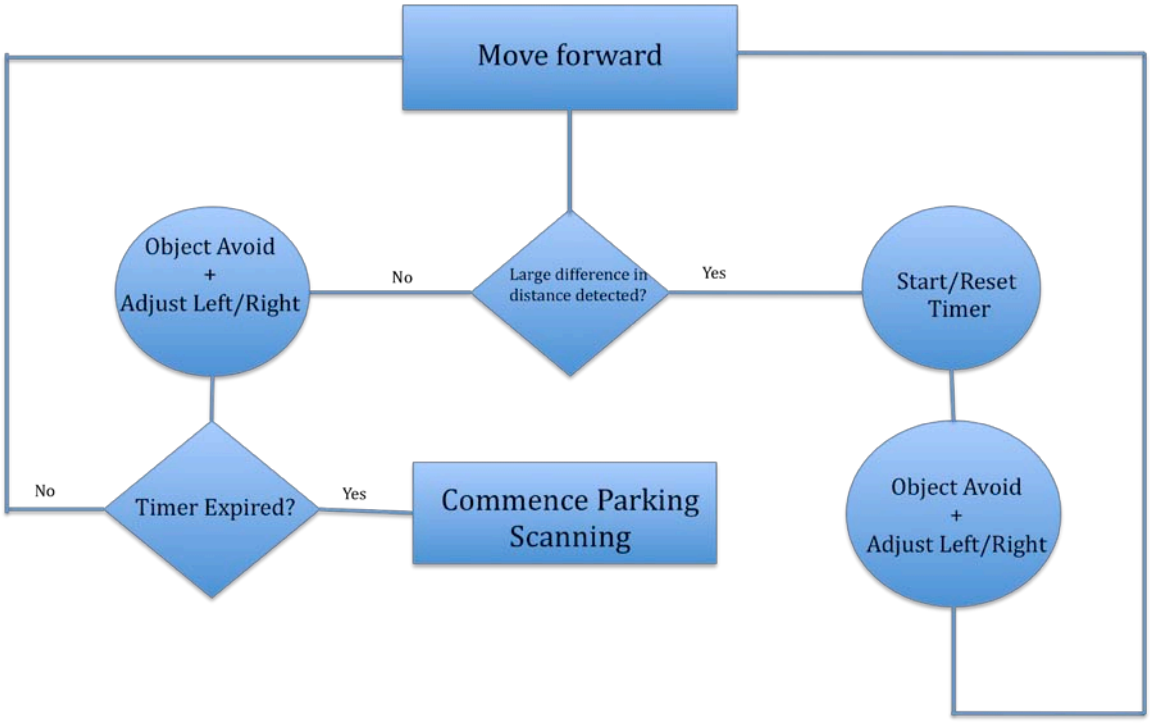

Figure 7.1 – Wall Following / Parking Seeking

This is the initial behavior of the Mobile Platform after the START message is sent from the VCU to the mobile platform. Figure 7.1 summarizes this behavior. The platform moves forward at a prescribed speed using the right-facing range finder to measure distance. As it moves forward it adjust left and right to keep a particular distance away from the wall or "cars". The wall following behavior uses two sets of values depending on whether the timer is ON or OFF, to compare the right-facing IR range finder A/D value to. This allows it to keep approximately the same distance away regardless if a "car" or just the wall is to the right of the robot (given that the autonomous platform initially starts the procedure next to a "car").

The object avoid behavior is integrated in the wall following. For the most part the mobile platform detects if it is going to collide with a car and performs immediate correction in course.

Once the timer has ran for long enough, it is determined that a parking space large enough for the mobile platform has been detected and behavior passes off to the Commence Parking Scanning Behavior. More detail can be found in the source code available in Appendix A – Mobile Platform.

Parking Scanning

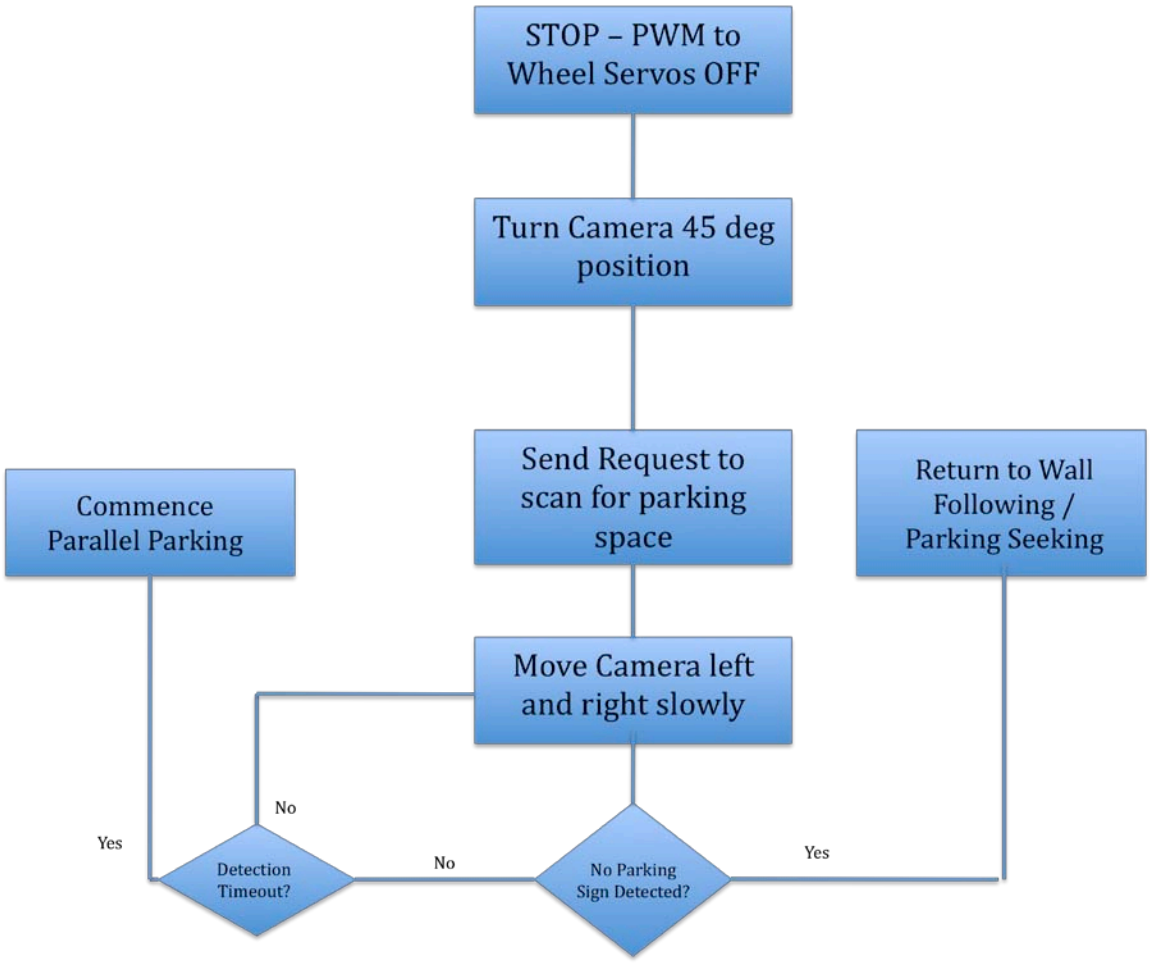

Figure 7.2 – Parking Scanning

This behavior is activated once a parking space has been detected from the Parking Seeking behavior. The mobile platform turns off the servos (in order to keep from possible drift because of hacked servos). Afterwards the Mobile Platform sets the initial position of the camera servo and sends a request to the VCU to begin scanning the space with the camera. The VCU commands the camera to be moved left and right until it can find the "No Parking" sign or after an amount to tries it times out and reply's to the Mobile Platform that no sign was detected. More detail can be found in the source code available in Appendix A – Mobile Platform.

Parking Scanning

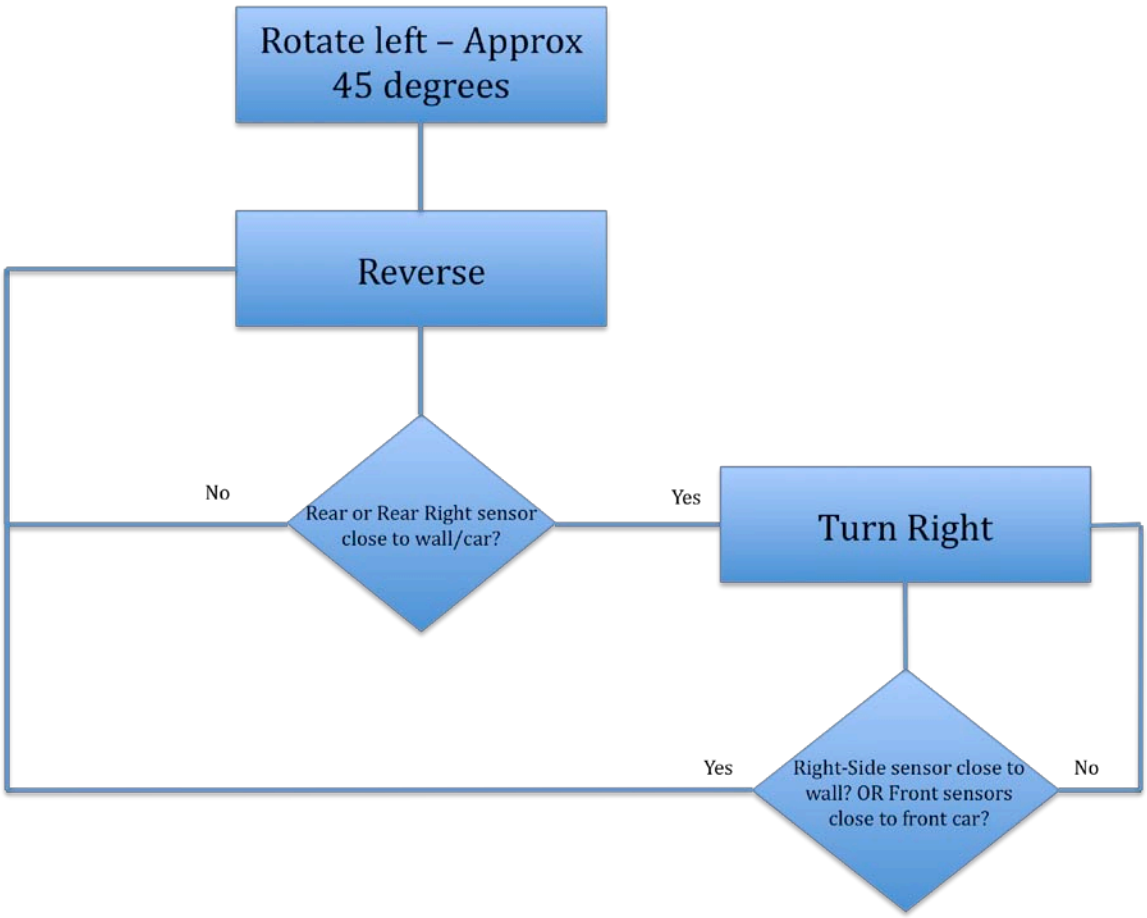

Figure 7.3 – Parallel Parking Behavior

The parallel parking behavior is quite simple. It relys on the rear sensors to to be a certain distance from the wall (quite close). Once the distance is reached the platform turns right until it cannot without striking and object in the front. Then given that there is space to reverse it will do so until it cannot anymore. A series of these actions leads to Parker-B successfully in the parking space. In the situation that this does not lead to a successful parking, the Mobile Platform has an attempt timeout in which the vehicle will turn left forward and reattempt to park again. More detail can be found in the source code available in Appendix A – Mobile Platform.

10.0 Conclusion

10.1 Work Accomplished

Parker-B is has a significant amount of work that was accomplished:

- Integration between two software systems
- Mobile platform design, construction and assembly
- Integration of sensors, communications, actuators and microcomputer.
- Parallel Parking behavior that mimics parallel parking
- Wall following behavior
- Computer vision

10.2 Limitations of Work

The behavior of Parker-B is impressive given the budgets constraints on project. Because of low cost IR range finders, it isn't guaranteed that Parker-B will detect a parking space or correctly follow walls 100% percent of the time.

The object detection, although quiet robust, only detects the parking sign because of its well defined "X" shape. If the sign was at a diagonal, the VCU wouldn't recognize it because it uses the symmetrical shape of the sign to recognize it. The VCU can only detect the sign in a range of angles as well so if the mobile platform looks at the sign at too large of an angle (with respect to the normal vector of the parking sign) it will not detect it.

Actuation on the Mobile platform is quiet imprecise as well. From test run to test run the servos may drift the platform left or right when the software intends on going straight or reverse. Significant amount of work went into wall following to correct for this error in actuation, however with low quality range finding with the IR range finders this made the processes difficult. Correcting imprecise actuation with imprecise sensors was the most testing part of this project because of the randomization of error.

Parallel Parking works quiet well, given that Parker-B wall follows correctly and leaves the mobile platform in a position where parallel parking is possible. However if the wall following sequence fails parallel parking will end in a disaster as well as parking space scanning. Parker-B may also nudge the car in front of it because of the large angle in movement in the front of the platform when Parker-B turns left or right. This however is only a small issue.

10.3 Technical Caveats

- When powering your camera off battery power, ensure the gauge in your wires is large enough for maximum current draw of the camera.

- Obtain or develop code to communicate with various protocols is early. This is critical for system integration on the project as a whole
- Turn PWM ports off when you want the robot to stay still. (For hacked servos only, because they may drift even when your software commands them to stop).
- Never skip on making weak electrical contacts; it may slow down progress during integration.

10.4 Future Work

The number one thing that would change if I could restart this project is to invest in better range finders (using sonar vs IR) and purchase motors; the amount of extra money on these things would have been well worth the trouble.

The servo's actuation was very unreliable when hacked. Movement by motors would have been preferred because of the smooth actuation. IR range finders are cheap, and easily influenced by the environment; things such as the color and reflectivity of surfaces can change output of the range finders.

11.0 References

[1] Seo, Naotoshi (2008, October), Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features), Available: March 2011 http://not.sonots.com/SciSoftware/haartraining.html

[2] Wikipedia (2010, October) Haar-like features Available: March 2011 http://en.wikipedia.org/wiki/Haar-like_features

12.0 Appendices

Appendix A – Mobile Platform

Main Code –

```
#include <avr/io.h>
#include "PVR.h"
#include "usart_driver.h"
#include <math.h>
#include <stdio.h>
#include <Math.h>

#define USART USARTF0
#define USART_BAUD 11500
#define SERIAL_UBBRVAL(baud) ((((F_CPU / 16) + (baud / 2)) / (baud)) - 1)
#define AVERGING_FILT 4
#define SAMPLE_SIZE 32
#define PARKING_SIZE 3100 //miliseconds
#define BASE_SPEED 17
#define ADJUST_SPEED 5
#define CAR_WIDTH 975
#define ATTEMPT_COUNT 100
#define WAIT_LOOPS 10000
```

```
USART_data_t USART_data;

/**
returns the distance in centimeters read by sensor 1
parameter: value Valid only between 4-7mm
**/

void turnLeft(int speed, int miliseconds)
{
      ServoD0(speed);
      ServoD1(0);
      delay_ms(miliseconds);
}

void turnRight(int speed, int miliseconds)
{
      ServoD1(-speed);
      ServoD0(0);
      delay_ms(miliseconds);
}

void reverse(int speed, int miliseconds)
{
      ServoD1(speed);
      ServoD0(-(speed-1));
      delay_ms(miliseconds);
}

void straight(int speed, int miliseconds)
{
      ServoD1(-1*speed);
      ServoD0(speed-1);
      delay_ms(miliseconds);
}


void commandServos(int leftSpeed, int rightSpeed)
{
      ServoD0(rightSpeed);
      ServoD1(-1*leftSpeed);
}

void stopServos()
{
      ServoD0(0);
      ServoD1(0);
      //deinitialize servos (keep from drifting when stopped)
      PORTD_DIR = 0x00;
}
unsigned int normalSensorDistance(int value)
{
      return (.000004*value*value - .026*value + 52.36)*10;
}

unsigned int shortRangeSensorDistance(int value)
{
      return (-.001*value+6.004)*10;
}

unsigned int getDistanceForward()
{
      int distanceLeft = 0;
      int x;
      for(x = 0; x < AVERGING_FILT; x++)
      {
        distanceLeft += ADCA2();
      }
      distanceLeft = distanceLeft >> 2;
      return distanceLeft;
```

```
}

unsigned int getDistanceRight()
{
      int distanceRight = 0;
      int x;
      for(x = 0; x < AVERGING_FILT; x++)
      {
        distanceRight += ADCA1();
      }
      distanceRight = distanceRight >> 2;
      return distanceRight;
}

unsigned int getDistanceRear()
{
      int distanceRear = 0;
      int x;
      for(x = 0; x < AVERGING_FILT; x++)
      {
        distanceRear += ADCA3();
      }
      distanceRear = distanceRear >> 2;
      return distanceRear;
}
unsigned int getSideDistance()
{
      int      averageDistance = 0;
      int x;
      for(x = 0; x < AVERGING_FILT; x++)
      {
        averageDistance += ADCA4();
      }
      averageDistance = averageDistance >> 2;
}

unsigned int getDistanceFront()
{
      int distanceFront = 0;
      int x;
      for(x = 0; x < AVERGING_FILT; x++)
      {
        distanceFront += ADCA0();
      }
      distanceFront = distanceFront >> 2;
      return distanceFront;
}

unsigned int getCarSensorDistance()
{
      int distanceSide = 0;
      int x;
      for(x = 0; x < AVERGING_FILT; x++)
      {
        distanceSide += ADCA5();
      }
      distanceSide = distanceSide >> 2;
      return distanceSide;
}
void usart_initialize(void)
{
      //pin 3 output
      PORTF.DIRSET = PIN3_bm;
      //pin2 input
      PORTF.DIRCLR = PIN2_bm;
//    USART_InterruptDriver_Initialize(&USART_data, &USART, USART_DREINTLVL(3));
      //usartc0, 8 data bits, no parity, 1 stop bit
      USART_Format_Set(&USART, USART_CHSIZE_8BIT_gc, USART_PMODE_DISABLED_gc, false);
      //ENABLE INTERRUPT
```

```
//    USART_RxdInterruptLevel_Set(USART_data.usart, USART_RXCINTLVL(3));
      //set baud rate
      USART_Baudrate_Set(&USART, 17, 0);

      //ENABLE RX AND TX
      USART_Rx_Enable(&USART);
      USART_Tx_Enable(&USART);
      //Enabel PMIC Interrupt level low
      //PMIC.CTRL |= PCMIC_LOLVLEX_bm;

      //enable global interrupts
      //sei();
}

inline void usart_tx_byte(char DataByte)
{
      int txrxVal = 0;

      while(1)
      {
        txrxVal = USARTF0_STATUS;
        txrxVal &= 0x20;;
        if(txrxVal == 0x20)
        {
                USARTF0_DATA = DataByte;
                break;
        }
      }
}

inline void usart_tx_string(char *StringPtr)
{
      int i = 0;
      while(StringPtr[i] != 0) //while not null terminator
      {
        usart_tx_byte(StringPtr[i]);
        i++;
      }
}


inline char usart_rx_byte(void)
{
        int txrxVal =  USARTF0_STATUS;
        txrxVal &= 0x80;
        char data;
        if(txrxVal == 0x80)
        {
                data = USARTF0_DATA;
        }
        else
        {
                data = 0;
        }
        return data;
}
/**
ISR(USARTC0_RXC_vect)
{
      USART_RXComplete(&USART_data);
}

ISR(USARTC0_DRE_vect)
{
      USART_DataRegEmpty(&USART_data);
}
**/
int main(void)
{
```

```
      xmegaInit();                                                    //setup XMega
      delayInit();                                                    //setup delay
functions
      ServoCInit();                                                   //setup PORTC Servos
      ServoDInit();                                                   //setup PORTD Servos
      ServoC0(-180);                                                         //camera
initial position
      ADCAInit();                                                             //setup PORTA
analong readings
      lcdInit();                                                              //setup LCD on
PORTK
      PORTQ_DIR |= 0x01;                                              //set Q0 (LED) as
output
      usart_initialize(); //setup usart c0

      //setup usart
      PORTC_OUT |= 0x08; //set pin 3 high
      PORTC_DIR |= 0X08; //set pin 3 as output
      USARTC0_CTRLC = 0x03; //asynchornous, no parity, 1 stop bid, 8bit char
      //BAUD APPROX = 19200
      USARTC0_BAUDCTRLA = 0X68;
      USARTC0_BAUDCTRLB = 0X00;
      USARTC0_CTRLA = 0X00; //disable usart interrupts
      USARTC0_CTRLC = 0X18; //enable transmitter, reciever

      //initialize timer
      initTimer();
      int i = 0;
      int averageDistance;
      int prevAverageDistance = 2000;
      char random;
      unsigned sampleCount = 0;
      unsigned int samples[SAMPLE_SIZE];
      int txrxVal;
      char data = 0;
      unsigned int averageSamples;
      unsigned int distanceMiddle;
      unsigned int distanceLeft;
      unsigned int distanceRight;
      unsigned int distanceRear;
      unsigned int distanceFront;
      unsigned int leftSpeed = BASE_SPEED;
      unsigned int rightSpeed = BASE_SPEED;
      unsigned int carSensorDistance;
      int waitLoops = WAIT_LOOPS;
      int cameraServo = 0;
      bool stop = false;
      bool timerStarted = false;
      bool parked = false;
      bool firstLoop = true;
      bool isIllegal;
      char distanceString[5] = {0, 0, 0, 0, 0};

      //wait for signal to start
      lcdString("Ready to Start!");
      while(data == 0)
      {
        data = usart_rx_byte();
      }
      while(true)
      {
       lcdGoto(0,0);
       lcdString("                ");
       lcdGoto(0,0);
        //Print out side IR range finder values
     lcdInt(averageDistance);
      lcdString(" ");
      lcdGoto(0,0);
      data = usart_rx_byte();
```

```
//Averaging filter
int x;
prevAverageDistance = averageDistance;
averageDistance = 0;
for(x = 0; x < AVERGING_FILT; x++)
{
  averageDistance += ADCA4();
}
averageDistance = averageDistance >> 2;
if(firstLoop)
{
  if(waitLoops >= WAIT_LOOPS)
  {
        firstLoop = false;
  }
  else
  {
        waitLoops++;
  }
}
if(stop)
{
  stopServos();
}
else
{
  if(!firstLoop)
  {
        commandServos(leftSpeed, rightSpeed);
  }
}
if(getDistanceForward() > 2000)
{
  reverse(10, 200);
  turnLeft(10, 100);
}
//if we're getting too close to right wall/ car!
if((averageDistance >= 4095 && !timerStarted && !firstLoop) ||
  (timerStarted && averageDistance > 1450 && !firstLoop))
{
  if(rightSpeed < BASE_SPEED + ADJUST_SPEED)
  {
        rightSpeed += ADJUST_SPEED;
        leftSpeed -= ADJUST_SPEED;
  }
  //about to hit car turn hard left
  if(getDistanceFront() > 3400)
  {
        //stopTimer();
        //timerStarted = false;
        turnLeft(10, 150);
  }
}
//too far from wall/car
else if((averageDistance < 3200 && !timerStarted && !firstLoop) ||
(averageDistance < 1375 && timerStarted && !firstLoop ))
{
  if(leftSpeed < BASE_SPEED + ADJUST_SPEED)
  {
        rightSpeed -= ADJUST_SPEED;
        leftSpeed += ADJUST_SPEED;
  }
}
//sweet spot
else if((averageDistance >= 3200 && !timerStarted) ||
  (timerStarted && averageDistance >= 1375 && averageDistance <= 1450))
{
  rightSpeed = BASE_SPEED;
```

```
  leftSpeed = BASE_SPEED;
}
//if we detected a large change in distance
if(fabs(prevAverageDistance - averageDistance) > CAR_WIDTH)
{
  if(!timerStarted)
  {
        startTimer();
        timerStarted = true;
        lcdGoto(1,0);
        lcdString("Flag");
        lcdGoto(0,0);
  }
  else if(getElapsedTime() > PARKING_SIZE)
  {
        lcdString("Parking Found");
        ServoC2(0); //move camera to face sideways
        stopServos();
        stopTimer();
        timerStarted = false;
        stop = true;
        lcdGoto(1,0);
        lcdString("Scanning Space...");
        //send request to vision software for sign detection
        usart_tx_byte('R');
        data = 0;
        while(data != 'T' && data != 'F')
        {
              data = usart_rx_byte();

              if(data == 'L')
              {
                    cameraServo += 2;
                    ServoC0(cameraServo);
              }
              else if(data == 'R')
              {
                    cameraServo -= 2;
                    ServoC0(cameraServo);
              }
        }
        if(data == 'T') //we found a no parking sign
        {
              lcdGoto(1,0);
              lcdString("Illegal Space!");
              lcdGoto(0,0);
              ServoDInit();
              ServoC0(-180);
              stop = false;
        }
        lcdGoto(0,0);
        ServoDInit();
        ServoC0(0);
  }
  //we must have past another "car" and the parkign space isn't big enough
  else
  {
        stopTimer();
        timerStarted = false;
        lcdGoto(1,0);
        lcdString("    ");
        lcdGoto(0,0);
  }
}
//if we detected a big change in distance, but then passed by a "taken" space
else if(averageDistance > 2100)
{
  if(timerStarted)
  {
```

```
        stopTimer();
        timerStarted = false;
        lcdGoto(1,0);
        lcdString("    ");
        lcdGoto(0,0);
  }
}

//if we're still driving past "empty parking space"
else if(averageDistance < 1600)
{
  if(timerStarted)
  {
        //if we went by enought "empty parking space to park
        int msPassed = getElapsedTime();
        if(msPassed > PARKING_SIZE)
        {
                lcdString("Parking Found");
                ServoC0(0); //move camera to face sideways
                stopServos();
                stopTimer();
                timerStarted = false;
                stop = true;
                lcdGoto(1,0);
                lcdString("Scanning Space...");
                //send request to vision software for sign detection
                usart_tx_byte('R');
                data = 0;
                while(data != 'T' && data != 'F')
                {
                        data = usart_rx_byte();
                        if(data == 'L')
                        {
                                cameraServo += 2;
                                ServoC0(cameraServo);
                        }
                        else if(data == 'R')
                        {
                                cameraServo -= 2;
                                ServoC0(cameraServo);
                        };
                }
                if(data == 'T') //we found a no parking sign
                {
                        lcdGoto(1,0);
                        lcdString("Illegal Space!");
                        lcdGoto(0,0);
                        ServoDInit();
                        ServoC0(0);
                        stop = false;
                }
                lcdGoto(0,0);
                ServoDInit();
                ServoC0(-180);
        }
  }

}

//begin parallel parking procedure
if(stop && !parked)
{
  /**
  //Hopefully, the rear IR can catch the wall or car
  distanceRear = 0;
  while(distanceRear <= 1250)
  {
        for(x = 0; x < AVERGING_FILT; x++)
        {
```

```
                distanceRear += ADCA3();
        }
        distanceRear = distanceRear >> 2;
        turnLeft(20, 100);
}
**/
//approx a 60 deg angle
turnLeft(15, 2000);
stopServos();
ServoDInit();
while(!parked)
{
        for(x = 0; x < AVERGING_FILT; x++)
        {
                distanceRear += ADCA3();
        }
        for(x = 0; x < AVERGING_FILT; x++)
        {
                distanceRight += ADCA1();
        }
        for(x = 0; x < AVERGING_FILT; x++)
        {
                distanceFront += ADCA0();
        }
        distanceRear = distanceRear >> 2;
        distanceRight = distanceRight >> 2;
        distanceFront = distanceFront >> 2;
        reverse(15, 100);

        distanceLeft = 0;
        for(x = 0; x < AVERGING_FILT; x++)
        {
                distanceLeft += ADCA4();
        }
        distanceLeft = distanceLeft >> 2;

        int rearDistance = 1500;
        bool done = false;
        int attemptCount = ATTEMPT_COUNT;
        //this sequence ensures the vehicle finds the corner of the space
        while(!done)
        {
                lcdGoto(1,0);
                lcdInt(getDistanceRear());
                lcdGoto(1,0);
                while(getDistanceRear() < 3600 && getDistanceRight() < 3500)
                {
                        reverse(10, 100);
                        distanceRear = getDistanceRear();
                }
                while(getDistanceRear() > rearDistance)
                {
                        turnRight(10,100);
                        distanceRear = getDistanceRear();
                        if(getSideDistance() > 3400 || getDistanceFront() > 2700 ||
                        getDistanceForward() > 2000)
                        {
                                break;
                        }

                        if(rearDistance < 4000)
                        {
                                rearDistance += 10;
                        }
                }
                if(getDistanceRear() > 3300 && getSideDistance() > 3400)
                {
                        done = true;
                }
```

```
                        else if(attemptCount == 0)
                        {
                                turnLeft(10, 2000);
                                straight(10, 1500);
                                turnLeft(10, 700);
                                attemptCount = ATTEMPT_COUNT;
                                rearDistance = 1500;
                        }
                        stopServos();
                        ServoDInit();
                        attemptCount--;


                }
                        stopServos();
                        ServoDInit();
                        lcdGoto(1,0);
                        lcdString("Parked");
                        parked = true;
                        //power off servos

          }

        }
        //Visual assurance of program running

        if (i >= 0)
        {
          i = -1;
          PORTQ_OUT = 1;                                //  turn on LED
        }
        else
        {
          PORTQ_OUT = 0;                                //  turn off LED
        //delay 100ms
          delay_ms(100);
          i = 1;
        }


        }
}
```

PVR.h

```
#ifndef  __PVR_h__
#define  __PVR_h__

#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"


#define LCD                     PORTK_OUT
#define LCDDDR          PORTK_DIR

volatile int delaycnt;

volatile int timercnt;

void xmegaInit(void);

void delayInit(void);

void delay_ms(int cnt);

void pvrdelay_us(int cnt);
```

```
void lcdDataWork(unsigned char c);
void lcdData(unsigned char c);
void lcdCharWork(unsigned char c);
void lcdChar(unsigned char c);
void lcdString(unsigned char ca[]);
void lcdInt(int value);
void lcdGoto(int row, int col);
void lcdInit(void);
void ServoCInit(void);
void ServoDInit(void);
void ServoC0(int value);
void ServoC1(int value);
void ServoC2(int value);
void ServoC3(int value);
void ServoC4(int value);
void ServoC5(int value);
void ServoD0(int value);
void ServoD1(int value);
void ServoD2(int value);
void ServoD3(int value);
void ServoD4(int value);
void ServoD5(int value);
void ADCAInit(void);
int ADCA0(void);
int ADCA1(void);
int ADCA2(void);
int ADCA3(void);
int ADCA4(void);
int ADCA5(void);
int ADCA6(void);
int ADCA7(void);
#endif
```

PVR.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"

/*********
 * Xmega *
 *********/

volatile int timercnt;

void xmegaInit(void)
{
	CCP = 0xD8;
	CLK_PSCTRL = 0x00;
	PORTQ_DIR = 0x01;
	//setup oscilllator
	OSC_CTRL = 0x02;                                        //enable 32MHz internal clock
	while ((OSC_STATUS & 0x02) == 0);              //wait for oscillator to be ready
	CCP = 0xD8;                                                     //write
signature to CCP
	CLK_CTRL = 0x01;                                        //select internal 32MHz RC
oscillator
}

/*********
 * Delay *
 *********/

void delayInit(void)
{
	TCF1_CTRLA = 0x01;                                              //set clock/1
	TCF1_CTRLB = 0x31;                                              //enable COMA and
COMB, set to FRQ
	TCF1_INTCTRLB = 0x00;                                   //turn off interrupts for
COMA and COMB
	SREG |= CPU_I_bm;                                               //enable all
interrupts
	PMIC_CTRL |= 0x01;                                              //enable all low
priority interrupts
}

void delay_ms(int cnt)
{
	delaycnt = 0;                                                   //set count value
	TCF1_CCA = 32000;                                               //set COMA to be 1ms
delay
	TCF1_CNT = 0;                                                   //reset counter
	TCF1_INTCTRLB = 0x01;                                   //enable low priority
interrupt for delay
	while (cnt != delaycnt);                        //delay
	TCF1_INTCTRLB = 0x00;                                   //disable interrupts
}

void pvrdelay_us(int cnt)
{
	delaycnt = 0;                                                   //set counter
	TCF1_CCA = 32;                                                  //set COMA to be 1us
delay
	TCF1_CNT = 0;                                                   //reset counter
	TCF1_INTCTRLB = 0x01;                                   //enable low priority
interrupt for delay
	while (cnt != delaycnt);                        //delay
	TCF1_INTCTRLB = 0x00;                                   //disable interrupts
}

void initTimer()
{
	TCC1_CTRLA = 0x01;                                              //set clock/1
	TCC1_CTRLB = 0x31;                                              //enable COMA and
```

```
COMB, set to FRQ
      TCC1_INTCTRLB = 0x00;                                  //turn off interrupts for
COMA and COMB
      SREG |= CPU_I_bm;                                             //enable all
interrupts
      PMIC_CTRL |= 0x01;
}
void startTimer()
{
      timercnt = 0;                                                 //set counter
      TCC1_CCA = 32000;                                                     //set COMA to
be 1ms delay
      TCC1_CNT = 0;                                                 //reset counter
      TCC1_INTCTRLB = 0x01;                                  //enable low priority
interrupt for delay
}

int getElapsedTime()
{
      return timercnt;
}

void stopTimer()
{
      TCC1_INTCTRLB = 0x00;                                  //disable interrupts
}


SIGNAL(TCC1_CCA_vect)
{
      timercnt++;
}

SIGNAL(TCF1_CCA_vect)
{
      delaycnt++;
}

SIGNAL(TCF0_CCA_vect)
{
      timercnt++;
}

/*******
 * LCD *
 *******/

#define LCD                   PORTK_OUT
#define LCDDDR        PORTK_DIR

void lcdDataWork(unsigned char c)
{
      c &= 0xF0;                                                            //keep data
bits, clear the rest
      c |= 0x08;                                                            //set E high
      LCD = c;                                                      //write to LCD
      delay_ms(2);                                                  //delay
      c ^= 0x08;                                                            //set E low
      LCD = c;                                                      //write to LCD
      delay_ms(2);                                                  //delay
      c |= 0x08;                                                            //set E high
      LCD = c;                                                      //write to LCD
      delay_ms(2);                                                  //delay
}

void lcdData(unsigned char c)
{
      unsigned char cHi = c & 0xF0;                  //give cHi the high 4 bits of c
      unsigned char cLo = c & 0x0F;                  //give cLo the low 4 bits of c
```

```
      cLo = cLo * 0x10;                                          //shift cLo left 4
bits
      lcdDataWork(cHi);
      lcdDataWork(cLo);
}

void lcdCharWork(unsigned char c)
{
      c &= 0xF0;                                                        //keep data
bits, clear the rest
      c |= 0x0A;                                                        //set E and RS
high
      LCD = c;                                                   //write to LCD
      delay_ms(2);                                               //delay
      c ^= 0x08;                                                        //set E low
      LCD = c;                                                   //write to LCD
      delay_ms(2);                                               //delay
      c |= 0x08;                                                        //set E high
      LCD = c;                                                   //write to LCD
      delay_ms(2);                                               //delay
}

void lcdChar(unsigned char c)
{
      unsigned char cHi = c & 0xF0;                  //give cHi the high 4 bits of c
      unsigned char cLo = c & 0x0F;                  //give cLo the low 4 bits of c
      cLo = cLo * 0x10;                                          //shift cLo left 4
bits
      lcdCharWork(cHi);
      lcdCharWork(cLo);
}


void lcdString(unsigned char ca[])
{
      int i = 0;
      while (ca[i] != '\0')
      {
        lcdChar(ca[i++]);
      }
}

void lcdInt(int value)
{
      int temp_val;
      int x = 10000;
      int leftZeros=5;

      if (value<0)
      {
           lcdChar('-');
         value *= -1;
      }

    while (value / x == 0)
    {
       x/=10;
       leftZeros--;
    }


    while ((value > 0) || (leftZeros>0))
    {
        temp_val = value / x;
        value -= temp_val * x;
        lcdChar(temp_val+ 0x30);
        x /= 10;
        leftZeros--;
    }
```

```
    while (leftZeros>0)
      {
        lcdChar(0+ 0x30);
      leftZeros--;
    }

    return;
}

void lcdGoto(int row, int col)
{
      unsigned char pos;
      if ((col >= 0 && col <= 19) && (row >= 0 && row <= 3))
      {
        pos = col;
        if (row == 1)
               pos += 0x40;
        else if (row == 2)
               pos += 0x14;
        else if (row == 3)
               pos += 0x54;
        lcdData(0x80 + pos);
      }
}

void lcdInit(void)
{
      delayInit();                                  //set up the  functions
      LCDDDR = 0xFF;                                //set LCD port to outputs.
      delay_ms(20);                                 //wait to ensure LCD powered up
      lcdDataWork(0x30);                            //put in 4 bit mode, part 1
      delay_ms(10);                                 //wait for lcd to finish
      lcdDataWork(0x30);                            //put in 4 bit mode, part 2
      delay_ms(2);                                  //wait for lcd to finish
      lcdData(0x32);                                //put in 4 bit mode, part 3
      lcdData(0x2C);                                //enable 2 line mode
      lcdData(0x0C);                                //turn everything on
      lcdData(0x01);                                //clear LCD
}

/*********
 * Servo *
 *********/

void ServoCInit(void)
{
      TCC0_CTRLA = 0x05;                            //set TCC0_CLK to CLK/64
      TCC0_CTRLB = 0xF3;                            //Enable OC A, B, C, and D.  Set to
Single Slope PWM
                                                            //OCnX = 1 from Bottom to CCx
and 0 from CCx to Top
      TCC0_PER = 10000;                             //20ms / (1/(32MHz/64)) = 10000.  PER
= Top
      TCC1_CTRLA = 0x05;                            //set TCC1_CLK to CLK/64
      TCC1_CTRLB = 0x33;                            //Enable OC A and B.  Set to Single
Slope PWM
                                                            //OCnX = 1 from Bottom to CCx
and 0 from CCx to Top
      TCC1_PER = 10000;                             //20ms / (1/(32MHz/64)) = 10000.  PER
= Top
      PORTC_DIR = 0x3F;                             //set PORTC5:0 to output
      TCC0_CCA = 0;                                 //PWMC0 off
      TCC0_CCB = 0;                                 //PWMC1 off
      TCC0_CCC = 0;                                 //PWMC2 off
      TCC0_CCD = 0;                                 //PWMC3 off
      TCC1_CCA = 0;                                 //PWMC4 off
      TCC1_CCB = 0;                                 //PWMC5 off
}
```

```
void ServoDInit(void)
{
      TCD0_CTRLA = 0x05;                        //set TCC0_CLK to CLK/64
      TCD0_CTRLB = 0xF3;                        //Enable OC A, B, C, and D.  Set to
Single Slope PWM
                                                      //OCnX = 1 from Bottom to CCx
and 0 from CCx to Top
      TCD0_PER = 10000;                         //20ms / (1/(32MHz/64)) = 10000.  PER
= Top
      TCD1_CTRLA = 0x05;                        //set TCC1_CLK to CLK/64
      TCD1_CTRLB = 0x33;                        //Enable OC A and B.  Set to Single
Slope PWM
                                                      //OCnX = 1 from Bottom to CCx
and 0 from CCx to Top
      TCD1_PER = 10000;                         //20ms / (1/(32MHz/64)) = 10000.  PER
= Top
      PORTD_DIR = 0x3F;                         //set PORTC5:0 to output
      TCD0_CCA = 0;                             //PWMC0 off
      TCD0_CCB = 0;                             //PWMC1 off
      TCD0_CCC = 0;                             //PWMC2 off
      TCD0_CCD = 0;                             //PWMC3 off
      TCD1_CCA = 0;                             //PWMC4 off
      TCD1_CCB = 0;                             //PWMC5 off
}

void ServoC0(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCC0_CCA = (750 + value);           //Generate PWM.
}

void ServoC1(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCC0_CCB = (750 + value);           //Generate PWM.
}

void ServoC2(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCC0_CCC = (750 + value);           //Generate PWM.
}

void ServoC3(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCC0_CCD = (750 + value);           //Generate PWM.
}
```

```
void ServoC4(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCC1_CCA = (750 + value);           //Generate PWM.
}

void ServoC5(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCC1_CCB = (750 + value);           //Generate PWM.
}

void ServoD0(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCD0_CCA = (750 + value);           //Generate PWM.
}

void ServoD1(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCD0_CCB = (750 + value);           //Generate PWM.
}

void ServoD2(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCD0_CCC = (750 + value);           //Generate PWM.
}

void ServoD3(int value)
{
      if (value > 100)                    //cap at +/- 100
        value = 100;                      //  -100 => 1ms
      else if (value < -100)              //  0    => 1.5ms
        value = -100;                     //  100  => 2ms
      value *= 5;                                 //multiply value by 2.5
      value /= 2;                                 //  new range +/- 250
      TCD0_CCD = (750 + value);           //Generate PWM.
}

void ServoD4(int value)
{
```

```
      if (value > 100)                      //cap at +/- 100
        value = 100;                        //  -100 => 1ms
      else if (value < -100)                //  0    => 1.5ms
        value = -100;                       //  100  => 2ms
      value *= 5;                                   //multiply value by 2.5
      value /= 2;                                   //  new range +/- 250
      TCD1_CCA = (750 + value);             //Generate PWM.
}

void ServoD5(int value)
{
      if (value > 100)                      //cap at +/- 100
        value = 100;                        //  -100 => 1ms
      else if (value < -100)                //  0    => 1.5ms
        value = -100;                       //  100  => 2ms
      value *= 5;                                   //multiply value by 2.5
      value /= 2;                                   //  new range +/- 250
      TCD1_CCB = (750 + value);             //Generate PWM.
}

/********
 * ADCA *
 ********/

void ADCAInit(void)
{
      delayInit();
      ADCA_CTRLB = 0x00;                            //12bit, right adjusted
      ADCA_REFCTRL = 0x10;                  //set to Vref = Vcc/1.6 = 2.0V (approx)
      ADCA_CH0_CTRL = 0x01;                 //set to single-ended
      ADCA_CH0_INTCTRL = 0x00;        //set flag at conversion complete.  Disable
interrupt
      ADCA_CH0_MUXCTRL = 0x08;        //set to Channel 1
      ADCA_PRESCALER = 0x03;                //set the speed to slow for higher accuracy
      ADCA_CTRLA |= 0x01;                           //Enable ADCA
}

int ADCA0(void)
{
      ADCA_CH0_MUXCTRL = 0x00;        //Set to Pin 0
      ADCA_CTRLA |= 0x04;                           //Start Conversion on ADCA Channel 0
      while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);   //wait for conversion to complete
      delay_ms(5);
      int value = ADCA_CH0_RES;             //grab result
      return value;                                 //return result
}

int ADCA1(void)
{
      ADCA_CH0_MUXCTRL = 0x08;        //Set to Pin 1
      ADCA_CTRLA |= 0x04;                           //Start Conversion on ADCA Channel 0
      while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);   //wait for conversion to complete
      delay_ms(5);
      int value = ADCA_CH0_RES;             //grab result
      return value;                                 //return result
}

int ADCA2(void)
{
      ADCA_CH0_MUXCTRL = 0x10;        //Set to Pin 2
      ADCA_CTRLA |= 0x04;                           //Start Conversion on ADCA Channel 0
      while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);   //wait for conversion to complete
      delay_ms(5);
      int value = ADCA_CH0_RES;             //grab result
      return value;                                 //return result
}

int ADCA3(void)
{
```

```
	ADCA_CH0_MUXCTRL = 0x18;	//Set to Pin 3
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA4(void)
{
	ADCA_CH0_MUXCTRL = 0x20;	//Set to Pin 4
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA5(void)
{
	ADCA_CH0_MUXCTRL = 0x28;	//Set to Pin 5
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA6(void)
{
	ADCA_CH0_MUXCTRL = 0x30;	//Set to Pin 6
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA7(void)
{
	ADCA_CH0_MUXCTRL = 0x38;	//Set to Pin 7
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}
```

usart_driver.h (downloaded from Atmel)

```
/* This file has been prepared for Doxygen automatic documentation generation.*/
/*! \file *********************************************************************
 *
 * \brief  XMEGA USART driver header file.
 *
 *      This file contains the function prototypes and enumerator definitions
 *      for various configuration parameters for the XMEGA USART driver.
 *
 *      The driver is not intended for size and/or speed critical code, since
 *      most functions are just a few lines of code, and the function call
 *      overhead would decrease code performance. The driver is intended for
 *      rapid prototyping and documentation purposes for getting started with
 *      the XMEGA ADC module.
 *
```

```
 *      For size and/or speed critical code, it is recommended to copy the
 *      function contents directly into your application instead of making
 *      a function call.
 *
 * \par Application note:
 *      AVR1307: Using the XMEGA USART
 *
 * \par Documentation
 *      For comprehensive code documentation, supported compilers, compiler
 *      settings and supported devices see readme.html
 *
 * \author
 *      Atmel Corporation: http://www.atmel.com \n
 *      Support email: avr@atmel.com
 *
 * $Revision: 1694 $
 * $Date: 2008-07-29 14:21:58 +0200 (ti, 29 jul 2008) $  \n
 *
 * Copyright (c) 2008, Atmel Corporation All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * 3. The name of ATMEL may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY AND
 * SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *****************************************************************************/
#ifndef USART_DRIVER_H
#define USART_DRIVER_H

#include "avr_compiler.h"

/* USART buffer defines. */

/* \brief  Receive buffer size: 2,4,8,16,32,64,128 or 256 bytes. */
#define USART_RX_BUFFER_SIZE 4
/* \brief Transmit buffer size: 2,4,8,16,32,64,128 or 256 bytes */
#define USART_TX_BUFFER_SIZE 4
/* \brief Receive buffer mask. */
#define USART_RX_BUFFER_MASK ( USART_RX_BUFFER_SIZE - 1 )
/* \brief Transmit buffer mask. */
#define USART_TX_BUFFER_MASK ( USART_TX_BUFFER_SIZE - 1 )


#if ( USART_RX_BUFFER_SIZE & USART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif
#if ( USART_TX_BUFFER_SIZE & USART_TX_BUFFER_MASK )
#error TX buffer size is not a power of 2
#endif
```

```
/* \brief USART transmit and receive ring buffer. */
typedef struct USART_Buffer
{
      /* \brief Receive buffer. */
      volatile uint8_t RX[USART_RX_BUFFER_SIZE];
      /* \brief Transmit buffer. */
      volatile uint8_t TX[USART_TX_BUFFER_SIZE];
      /* \brief Receive buffer head. */
      volatile uint8_t RX_Head;
      /* \brief Receive buffer tail. */
      volatile uint8_t RX_Tail;
      /* \brief Transmit buffer head. */
      volatile uint8_t TX_Head;
      /* \brief Transmit buffer tail. */
      volatile uint8_t TX_Tail;
} USART_Buffer_t;


/*! \brief Struct used when interrupt driven driver is used.
*
*  Struct containing pointer to a usart, a buffer and a location to store Data
*  register interrupt level temporary.
*/
typedef struct Usart_and_buffer
{
      /* \brief Pointer to USART module to use. */
      USART_t * usart;
      /* \brief Data register empty interrupt level. */
      USART_DREINTLVL_t dreIntLevel;
      /* \brief Data buffer. */
      USART_Buffer_t buffer;
} USART_data_t;


/* Macros. */

/*! \brief Macro that sets the USART frame format.
 *
 *  Sets the frame format, Frame Size, parity mode and number of stop bits.
 *
 *  \param _usart        Pointer to the USART module
 *  \param _charSize     The character size. Use USART_CHSIZE_t type.
 *  \param _parityMode   The parity Mode. Use USART_PMODE_t type.
 *  \param _twoStopBits  Enable two stop bit mode. Use bool type.
 */
#define USART_Format_Set(_usart, _charSize, _parityMode, _twoStopBits)         \
      (_usart)->CTRLC = (uint8_t) _charSize | _parityMode |                      \
                        (_twoStopBits ? USART_SBMODE_bm : 0)


/*! \brief Set USART baud rate.
 *
 *  Sets the USART's baud rate register.
 *
 *  UBRR_Value   : Value written to UBRR
 *  ScaleFactor  : Time Base Generator Scale Factor
 *
 *  Equation for calculation of BSEL value in asynchronous normal speed mode:
 *    If ScaleFactor >= 0
 *      BSEL = ((I/O clock frequency)/(2^(ScaleFactor)*16*Baudrate))-1
 *    If ScaleFactor < 0
 *      BSEL = (1/(2^(ScaleFactor)*16))*(((I/O clock frequency)/Baudrate)-1)
 *
 *    \note See XMEGA manual for equations for calculation of BSEL value in other
 *        modes.
 *
 *  \param _usart          Pointer to the USART module.
 *  \param _bselValue      Value to write to BSEL part of Baud control register.
 *                         Use uint16_t type.
```

```
 *  \param _bScaleFactor   USART baud rate scale factor.
 *                         Use uint8_t type
 */
#define USART_Baudrate_Set(_usart, _bselValue, _bScaleFactor)                \
      (_usart)->BAUDCTRLA =(uint8_t)_bselValue;
\
      (_usart)->BAUDCTRLB =(_bScaleFactor << USART_BSCALE0_bp)|(_bselValue >> 8)


/*! \brief Enable USART receiver.
 *
 *  \param _usart    Pointer to the USART module
 */
#define USART_Rx_Enable(_usart) ((_usart)->CTRLB |= USART_RXEN_bm)


/*! \brief Disable USART receiver.
 *
 *  \param _usart Pointer to the USART module.
 */
#define USART_Rx_Disable(_usart) ((_usart)->CTRLB &= ~USART_RXEN_bm)


/*! \brief Enable USART transmitter.
 *
 *  \param _usart Pointer to the USART module.
 */
#define USART_Tx_Enable(_usart)      ((_usart)->CTRLB |= USART_TXEN_bm)


/*! \brief Disable USART transmitter.
 *
 *  \param _usart Pointer to the USART module.
 */
#define USART_Tx_Disable(_usart) ((_usart)->CTRLB &= ~USART_TXEN_bm)


/*! \brief Set USART RXD interrupt level.
 *
 *  Sets the interrupt level on RX Complete interrupt.
 *
 *  \param _usart        Pointer to the USART module.
 *  \param _rxdIntLevel  Interrupt level of the RXD interrupt.
 *                       Use USART_RXCINTLVL_t type.
 */
#define USART_RxdInterruptLevel_Set(_usart, _rxdIntLevel)                    \
      ((_usart)->CTRLA = ((_usart)->CTRLA & ~USART_RXCINTLVL_gm) | _rxdIntLevel)


/*! \brief Set USART TXD interrupt level.
 *
 *  Sets the interrupt level on TX Complete interrupt.
 *
 *  \param _usart        Pointer to the USART module.
 *  \param _txdIntLevel  Interrupt level of the TXD interrupt.
 *                       Use USART_TXCINTLVL_t type.
 */
#define USART_TxdInterruptLevel_Set(_usart, _txdIntLevel)                    \
      (_usart)->CTRLA = ((_usart)->CTRLA & ~USART_TXCINTLVL_gm) | _txdIntLevel



/*! \brief Set USART DRE interrupt level.
 *
 *  Sets the interrupt level on Data Register interrupt.
 *
 *  \param _usart        Pointer to the USART module.
 *  \param _dreIntLevel  Interrupt level of the DRE interrupt.
 *                       Use USART_DREINTLVL_t type.
```

```
 */
#define USART_DreInterruptLevel_Set(_usart, _dreIntLevel)                       \
      (_usart)->CTRLA = ((_usart)->CTRLA & ~USART_DREINTLVL_gm) | _dreIntLevel


/*! \brief Set the mode the USART run in.
 *
 * Set the mode the USART run in. The default mode is asynchronous mode.
 *
 *  \param  _usart       Pointer to the USART module register section.
 *  \param  _usartMode   Selects the USART mode. Use  USART_CMODE_t type.
 *
 *  USART modes:
 *  - 0x0        : Asynchronous mode.
 *  - 0x1        : Synchronous mode.
 *  - 0x2        : IrDA mode.
 *  - 0x3        : Master SPI mode.
 */
#define USART_SetMode(_usart, _usartMode)                                       \
      ((_usart)->CTRLC = ((_usart)->CTRLC & (~USART_CMODE_gm)) | _usartMode)



/*! \brief Check if data register empty flag is set.
 *
 *  \param _usart      The USART module.
 */
#define USART_IsTXDataRegisterEmpty(_usart) (((_usart)->STATUS & USART_DREIF_bm) != 0)



/*! \brief Put data (5-8 bit character).
 *
 *  Use the macro USART_IsTXDataRegisterEmpty before using this function to
 *  put data to the TX register.
 *
 *  \param _usart      The USART module.
 *  \param _data       The data to send.
 */
#define USART_PutChar(_usart, _data) ((_usart)->DATA = _data)



/*! \brief Checks if the RX complete interrupt flag is set.
 *
 *   Checks if the RX complete interrupt flag is set.
 *
 *  \param _usart     The USART module.
 */
#define USART_IsRXComplete(_usart) (((_usart)->STATUS & USART_RXCIF_bm) != 0)



/*! \brief Get received data (5-8 bit character).
 *
 *  This macro reads out the RX register.
 *  Use the macro USART_RX_Complete to check if anything is received.
 *
 *  \param _usart     The USART module.
 *
 *  \retval           Received data.
 */
#define USART_GetChar(_usart)  ((_usart)->DATA)


/* Functions for interrupt driven driver. */
void USART_InterruptDriver_Initialize(USART_data_t * usart_data,
                                      USART_t * usart,
```

```
                                    USART_DREINTLVL_t dreIntLevel );

void USART_InterruptDriver_DreInterruptLevel_Set(USART_data_t * usart_data,
USART_DREINTLVL_t dreIntLevel);

bool USART_TXBuffer_FreeSpace(USART_data_t * usart_data);
bool USART_TXBuffer_PutByte(USART_data_t * usart_data, uint8_t data);
bool USART_RXBufferData_Available(USART_data_t * usart_data);
uint8_t USART_RXBuffer_GetByte(USART_data_t * usart_data);
bool USART_RXComplete(USART_data_t * usart_data);
void USART_DataRegEmpty(USART_data_t * usart_data);

/* Functions for polled driver. */
void USART_NineBits_PutChar(USART_t * usart, uint16_t data);
uint16_t USART_NineBits_GetChar(USART_t * usart);

void UsartF0_Init(void);
void UsartPutChar(uint8_t sendData);


#endif
```

Appendix B - VCU Software

tracker.cpp

```
#include <opencv.hpp>
#include <cv.h>
#include <ml.h>
#include <cxcore.h>
#include <highgui.h>
#include <cassert>
#include <iostream>
#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */

#define NUMBER_ATTEMPTS 550
char  * WINDOW_NAME  = "Parker-b Cam";
char  * CONNECTION_NAME = "/dev/tty.RN42-1F80-SPP";
char  * CAMERA_URL = "http://192.168.1.105/img/video.mjpeg";
//const CFIndex CASCADE_NAME_LEN = 2048;
char *   CASCADE_NAME =
"/Users/Steve/Documents/IMDL/ParkingSignTraining/Cascade/parkingsigncascade14.xml";

using namespace std;

/**
 This application displays Parker-B's camera on the screen, identifys the objects and
commands Parker-B
 to follow the general direction of the face.
 **/

/*
 * 'open_port()' - Open serial port 1.
 *
 * Returns the file descriptor on success or -1 on error.
 */

int open_port(char * pPortName)
{
      int fd; /* File descriptor for the port */


      fd = open(pPortName, O_RDWR | O_NOCTTY | O_NDELAY);
```

```
      if (fd == -1)
      {
        /*
         * Could not open the port.
         */

        perror("open_port: Unable to open port");
      }
      else
        fcntl(fd, F_SETFL, 0);

      return (fd);
}

int main (int argc, char * const argv[])
{
    const int scale = 2;
      //open the bluetooth serial to Parker-B
      int fd = open_port(CONNECTION_NAME);
      bool leftRight = false;

    // create all necessary instances
    cvNamedWindow (WINDOW_NAME, CV_WINDOW_AUTOSIZE);
    CvCapture *camera = cvCreateFileCapture(CAMERA_URL);
      //CvCapture * camera = cvCreateCameraCapture (CV_CAP_ANY);
      //CvCapture *camera = cvCaptureFromCAM(0);
    CvHaarClassifierCascade* cascade = (CvHaarClassifierCascade*) cvLoad (CASCADE_NAME,
0, 0, 0);
    CvMemStorage* storage = cvCreateMemStorage(0);
    assert (storage);

    // you do own an iSight, don't you ?!?
    if (! camera)
        abort();

    // did we load the cascade?!?
    if (! cascade)
        abort ();

      //instruct Parker-b to commence parking seeking
      write(fd,"S",1);

      //Wait for Parking Sign detection request
      char request[1];
      fcntl(fd, F_SETFL, FNDELAY);
      //int size = read(fd, request, 1);

    // get an initial frame and duplicate it for later work
    IplImage *  current_frame = cvQueryFrame (camera);

    IplImage *  draw_image    = cvCreateImage(cvSize (current_frame->width,
current_frame->height), IPL_DEPTH_8U, 3);
    IplImage *  gray_image    = cvCreateImage(cvSize (current_frame->width,
current_frame->height), IPL_DEPTH_8U, 1);
    IplImage *  small_image   = cvCreateImage(cvSize (current_frame->width / scale,
current_frame->height / scale), IPL_DEPTH_8U, 1);
    assert (current_frame && gray_image && draw_image);


    // as long as there are images ...
      int imageCount = 0;
        bool detection = false;
      bool left = false;
      int turnDelay = 0;
      int CAMERA_RANGE = 50;
      int attempts = 0;
      float cameraAngle = 0;

      while(true)
```

```
	{

	  read(fd, request, 1);

	  current_frame = cvQueryFrame (camera);

	  if (attempts < NUMBER_ATTEMPTS )
	  {
		//theres a scanning request - so scan the space w/ camera
		if(request[0] == 'R')
		{
			if (left) {
				//turn camera left
				write(fd, "L", 1);
				cameraAngle += 1;
				if(cameraAngle > CAMERA_RANGE)
				{
					left = false;
				}
				turnDelay = 0;
			}
			else
			{
				//turn camera right
				write(fd, "R", 1);
				cameraAngle -= 1;
				if (cameraAngle <= 0)
				{
					left = true;
				}
				turnDelay = 0;
			}

		}
		// convert to gray and downsize
		cvCvtColor (current_frame, gray_image, CV_BGR2GRAY);
		cvResize (gray_image, small_image, CV_INTER_LINEAR);

		// detect
		CvSeq* objects = cvHaarDetectObjects (small_image, cascade, storage,
                                                      1.1, 2,
CV_HAAR_DO_CANNY_PRUNING,
                                                      cvSize (30, 30));

		// draw detections
		cvFlip (current_frame, draw_image, 1);
		for (int i = 0; i < (objects ? objects->total : 0); i++)
		{
			if(detection == false)
			{
				detection = true;
			}
			CvRect* r = (CvRect*) cvGetSeqElem (objects, i);
			CvPoint center;
			int radius;
			center.x = cvRound((small_image->width - r->width*0.5 - r->x)
*scale);
			center.y = cvRound((r->y + r->height*0.5)*scale);
			radius = cvRound((r->width + r->height)*0.25*scale);
			cvCircle (draw_image, center, radius, CV_RGB(0,255,0), 3, 8, 0 );
		}
		cvShowImage (WINDOW_NAME, draw_image);

		//parking sign detected
		if(detection && request[0] == 'R')
		{
			write(fd, "T", 1);
			request[0] = ' '; //erase request notification
		}
```

```
                // wait a tenth of a second for keypress and window drawing
                int key = cvWaitKey (1);
                if (key == 'q' || key == 'Q')
                        break;
                detection = false;
                attempts++;
        }
        else
        {
                //no parking sign detected
                if(attempts >= NUMBER_ATTEMPTS)
                {
                        write(fd, "F", 1);
                }
                attempts = 0;
                request[0] = ' ';
                detection = false;
        }



      }

    // be nice and return no error
    return 0;
}
```