Portland State University

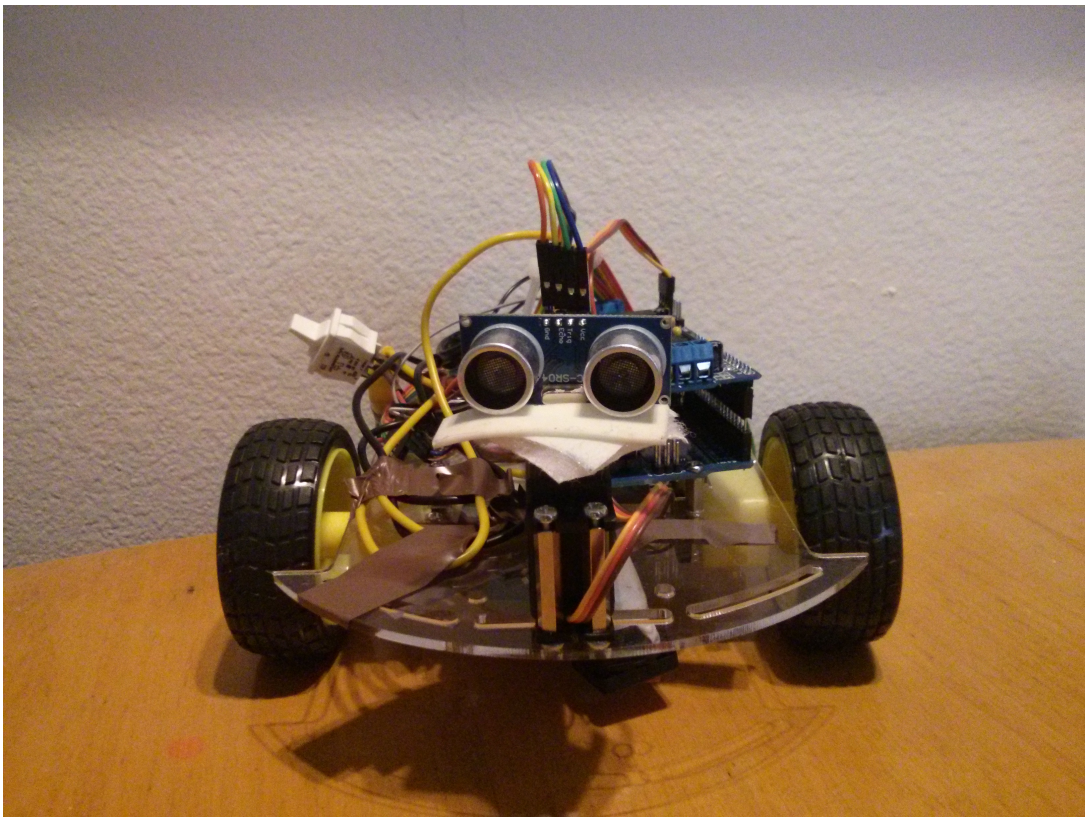
Embedded Systems on FPGAs

ECE544

Autonomous Robot Car

Erik Rhodes Caren Zgheib

June 9, 2014



Contents

1 Introduction

We created an autonomous robot car (SaMMY) that can successfully navigate its way through a given area without hitting any objects . It uses the popular **Arduino** platform and libraries along with a ultrasonic sensor for object detection.

2 Hardware Components

There are many different components needed to successfully and robustly create an autonomous robot. The bulk of our design used a robot kit we ordered online (ref). We also added various parts to make the production time more convenient, including switches, additional batteries, and various cables. The important components can be seen in Figure 1 and are described in Table 1.

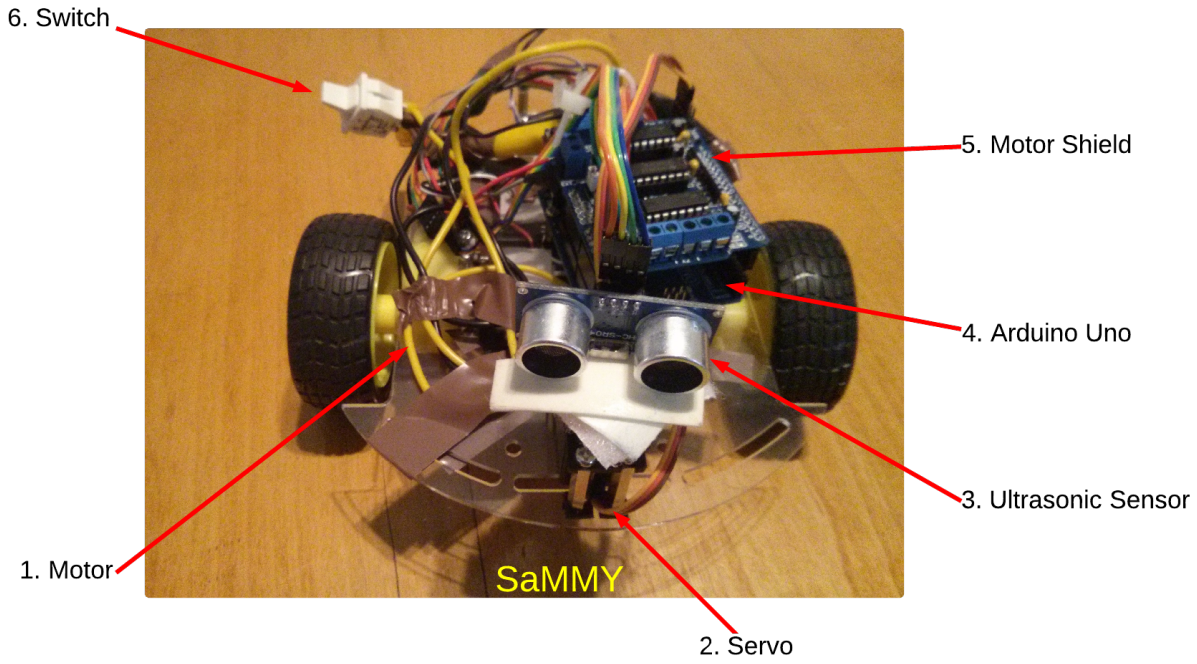


Figure 1: Block Diagram of Control System

Part Name	Function	Details
1. Motor	Drives wheels	N/A
2. RadioShack® Standard Servo	180°rotation	`http://shack.net/Ty2VB4`
3. Ultrasonic Ranging Module HC-SR04	Object detection	`http://bit.ly/1lhqwjI`
4. Arduino Uno	Microcontroller Board	`http://bit.ly/1gBB1NG`
5. L293D Motor Driver Shield	Controls motors	`http://bit.ly/1uILISF`
6. Switch	On/Off power button	N/A
7. Batteries (not pictured)	9V AA External Power	For Arduino/motors/servo

Table 1: Hardware Components List

Assembling the kit took a fair amount of time. We added a switch, soldered pins onto the motor shield for the sonar sensor, added the motor shield, changed the weight a few different times, and taped the battery holder to ensure the batteries wouldn't get displaced.

3 Robot Control

3.1 Functions

The software portion was written in the **Arduino** IDE, and we modified some of the existing libraries. The main functions for our project are listed in Table 2.

Component	Function	Details
Motor	Drive Forward	Both wheels moving forward at predefined speed
	Drive Backward	Both wheels moving backward at predefined speed
	Rotate Left/Right	Both wheels spin for "in-place" turn of 90°
	Veer Left/Right	Only outside wheel turns and vehicle does not stop
	U-Turn	Backs up then turns 180°
	Coolness	After certain time, bot does various spins
Sonar + Servo	Look Forward	Scans a 40°range in front while moving forward
	Look Left/Right	Scans 90°left/right to determine next turn

Table 2: Robot Functions

3.2 Algorithm

The primary purpose of our program is to ensure that SaMMY avoids all obstacles. Figure 2 shows how our code is structured, and Listing 1 provides a code snippet from the main loop. Once the setup has run and the components have been initialized, the program enters the main section of our program and runs in an infinite loop which scans for objects and makes decisions on which direction to go. First, the `scanClear()` function is called, which checks for objects in front of the robot within a 40° angle width. If this value is *True*, the area is clear and the robot drives forward. Otherwise, it stops, calls `lookLeft()` and `lookRight()` and calculates the average distance to each side. If `distanceLeft` is greater than `distanceRight` and greater than `dangerThreshold`, the bot backs up and rotates left. Likewise, if `distanceRight` is greater than `distanceLeft` and greater than the `dangerThreshold`, the bot backs up and rotates right. Otherwise, it is blocked on both sides. At this point it backs up and makes a U-turn.

To add some character to our robot, we added a counter that would launch SaMMY into a "victory dance". We check this value before launching the `scanClear()` function. If it's time to dance, then the `coolness()` function is called and SaMMY performs his victory dance.

Figure 3 shows a more detailed flowchart of the algorithm for `scanClear()`, which is called at the start of every loop.

This function calls `lookForward()`, which returns 5 readings at 10° increments. If any of these distances is less than `COLLISION_DISTANCE`, the `clear` flag is set to *FALSE* and `EMERGENCY_DISTANCE` is checked. If the value is below the emergency distance, the program immediately returns *FALSE* so that the appropriate action can be taken. Otherwise, the program function finishes its analysis and then returns the `clear` value. Provided the `clear` flag is *TRUE*, the robot then checks for the

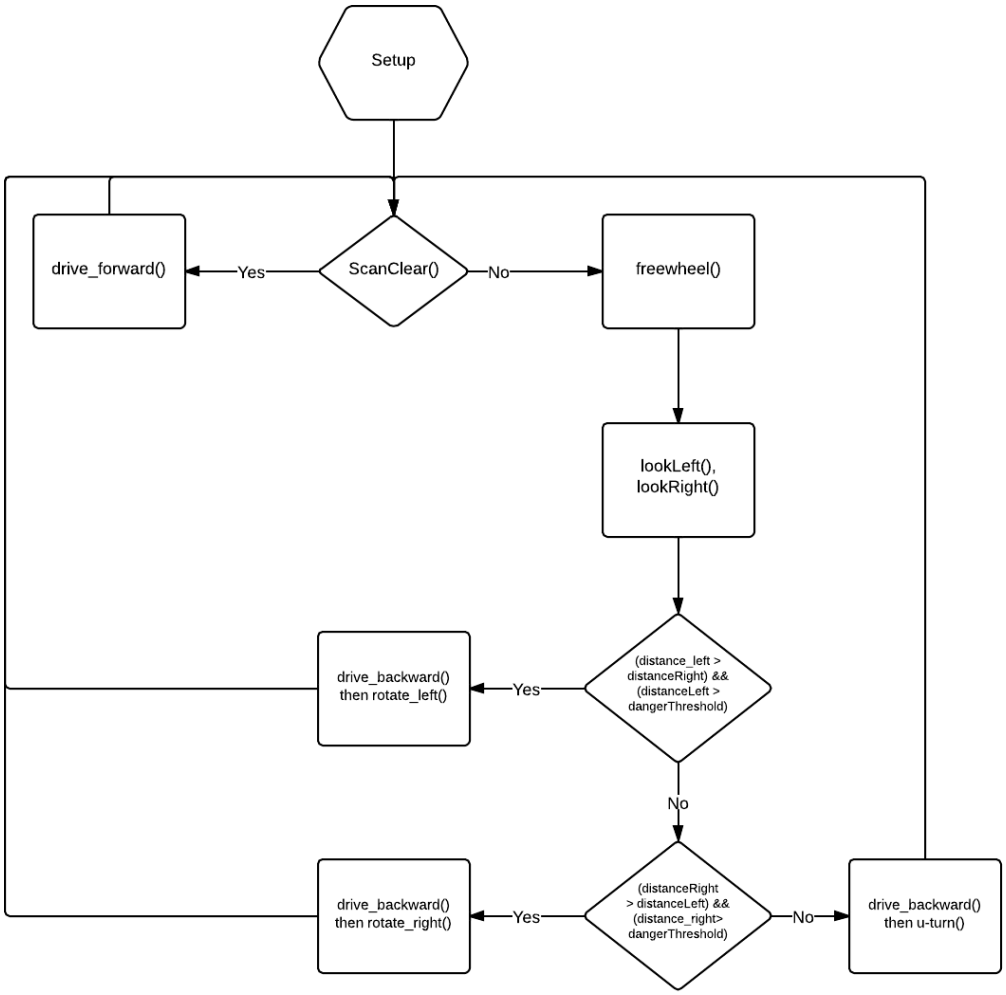


Figure 2: Main Loop Flowchart

average distance on the left and right. If exactly one side's distance is less than `AVOIDANCE_DISTANCE` (the first and therefore farthest distance the bot checks) it veers the opposite way and continues driving forward. Once `scanClear()` has finished, it returns the `clear` value and the next actions are taken (Figure 2).

The `scan()` function actually performs the scans at each angle by sending a sonar "ping" and waiting for the return signal to measure the distance. Sometimes the sonar returns very low, erroneous values that would cause the robot to halt. To avoid that, the `scan()` function only stops the bot and re-centers the sonar if two ping results in a row are below the emergency distance.

Listing 1: Control Loop

```
1 void loop() {
2   if (scanClear())  drive_forward();
3   else              // if path is blocked
4   {
5     freewheel();    // stop
6     lookLeft();
7     int distanceLeft = getAverageDistance();
```

```
 8
 9       lookRight();
10       int distanceRight = getAverageDistance();
11
12       servo_position(CENTER);
13
14       // go the least obstructed way
15       if (distanceLeft > distanceRight && distanceLeft > dangerThreshold)
16       {
17         drive_backward();
18         delay(400);
19         rotate_left();
20       }
21       else if (distanceRight > distanceLeft && distanceRight > dangerThreshold)
22       {
23         drive_backward();
24         delay(400);
25         rotate_right();
26       }
27       else     // equally blocked or less than danger threshold left or right
28       {
29         freewheel();
30         delay(20);
31         drive_backward();
32         delay(500);
33         u_turn();
34       }
35     }
36 }
37 }
```

4 Conclusion

`Servo and Protecto` was quite fun for us to build and was certainly the biggest "crowd pleaser" out of all the projects we've made. The assembly perhaps took the least amount of time, although we did face problems with a few of the parts not being built correctly (mainly the battery holder and motor plate mount). Fixing the power problems was the most frustrating part of the project, but once we finished everything, the final product was quite rewarding. We both spent equal times on all the different aspects of this project. Currently, SaMMY performs quite well, even if he does run into walls occasionally. This is caused by veering, which was a tough issue to fix. Listed below are some of the challenges we faced.

4.1 Challenges

- **Power Issues**: After the initial assembly, we were not able to properly move the servo and control the motors. Our initial thought was inadequate power, but the correspondent we communicated with ensured us that it could run perfectly fine on 6V of AA batteries. Even after upping the voltage we saw problems, which were due to the power demands of our servo. After increasing the power to 9V of AA batteries the problem diminished.

- **Motor control**: The motors we received are not precise enough to deliver equal power on both sides. This caused inevitable veering of the robot. We were able to use our own veering function to correct a lot of this, but were not able to control the bot perfectly.

- **CMUcam**: We were able to calibrate and test the CMUcam, which was intended for color detection. However, communicating the frames from the camera to the Arduino proved difficult. Even the example code would not work on our device. We were not able to solve the communication error before the project was due.
- **Erroneous Sonar Readings**: Occasionally the ultrasonic sensor picked up incorrect values distance values that would immediately cause our bot to stop (possibly due to ghost echoes). We adjusted this by waiting until it received two pings below 10cm.

4.2 Future Additions

The Arduino platform and components in this project provide a large amount of functionality. Because of this, the amount of improvements is virtually limitless. Our goals in the future could involve adding:

- **Battery Pack**: Buying a powerful, rechargeable battery pack would save money, improve performance and decrease hassle.
- **External Apparatus**: We initially wanted to attach a simple device that could be triggered based on certain events. While launching or shooting an object would have been interesting, in most cases this would require PID algorithms, which would involve sufficiently more work and tuning.
- **Color Detection**: One of our original ideas (and part of the reason for the name **Servo and Protecto**) was to use color detection to differentiate between "friends" and "enemies". There would be certain objects that the robot would purposely avoid, and others that it would just run over.
- **Infrared**: Infrared sensors can be used to detect edges, and they could also be used as an external controller/remote.
- **Object Following**: The sonar and/or color detector could be used to follow an object instead of avoid them. This would be useful when if we attached a physical arm or apparatus of some sort.

5 References

Source	Description	Link
Arduino Libraries	Servo	`http://bit.ly/1nuXP2Y`
	Sonar	`http://bit.ly/1pYs3OV`
	Motor	`http://bit.ly/1pepI2n`
Oddwires Robot Kit	Kit Description	`http://bit.ly/TysmCv`
	Assembly Instructions	`http://bit.ly/1xASGf2`

Table 3: Robot Functions

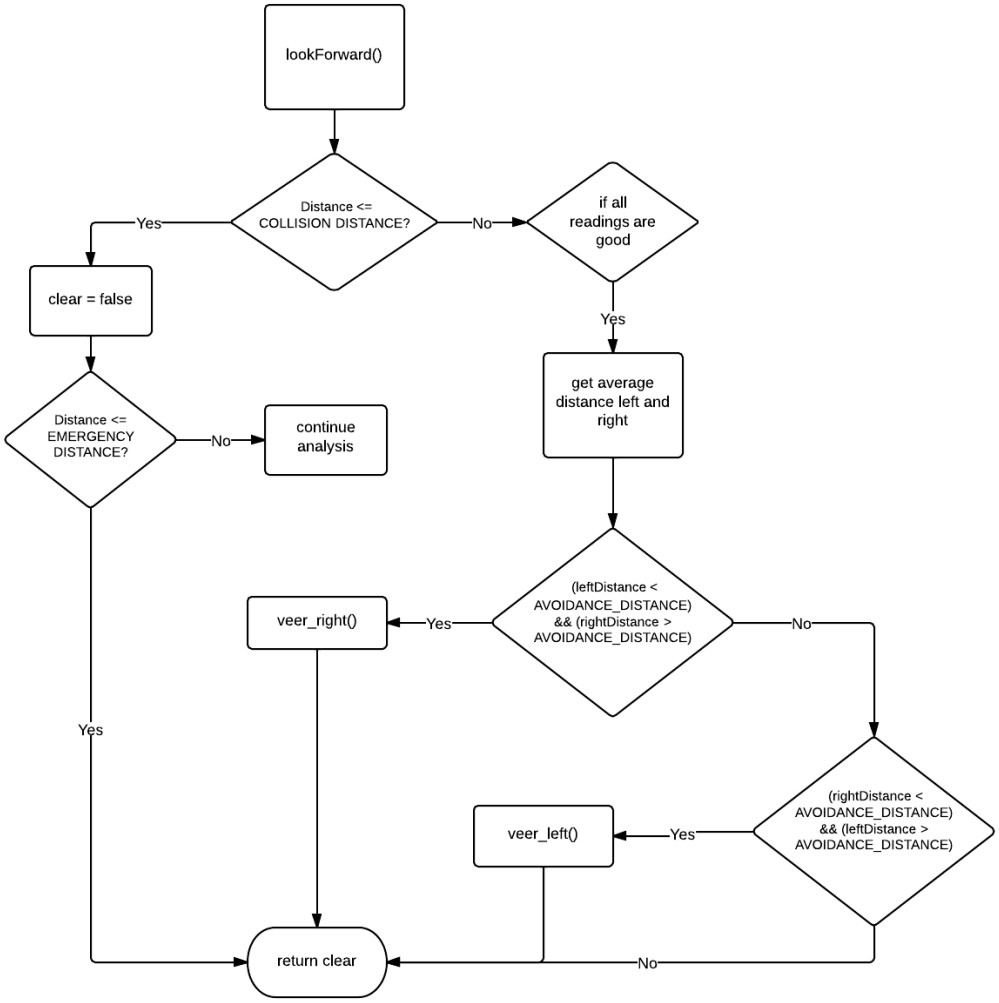


Figure 3: scanClear() Flowchart