Contents

Project

Here is the main page for my robotics project. I will add sub-pages as I complete various project stages. I am excited to get started and see what ideas I come up with. Let me know what you guys think with some comments.

Project Idea

Project Parts

Project Design

Project Implementation

Project Results I - Integration and Testing

Project Results II - Discussion and Analysis

Project Future Work

So that's it... project over. I had fun and learned a lot. Hopefully anyone reading this can also learn from my experience and my mistakes.

Here's the playlist of all 4 project videos in case you missed one!

YOUTUBE

Project Idea

My initial idea for this robotics project was an idea I had when first reading about the Arduino at the beginning of this course. There is a quick mention in the history of Arduino about a project one student had that had an alarm clock moving up on a string or something to get a person to wake up. My idea was to have a robotic alarm clock that runs away and hides to have the user wake up and have to shut off the alarm. I searched around on the Internet and I found that someone has already come up with this idea. The robot is called "Clocky" and is basically the exact idea I had. However, I looked at the product on Amazon, and noticed that it had some shortcomings. One of the main problems users had was that the robot did not move around much because it would either not move well on a carpet or get stuck on objects (as seen in the video below). Therefore, I plan to forge ahead with my initial idea, hopefully improving the functionality of this product so that the alarm is more effective by having the robot implement a better form of movement functionality.

https://youtu.be/o2rUhCsHcZ0

Since the exact details of the control architecture for the Clocky are unavailable, and it does not work effectively anyway, I will start from scratch in designing my robot. The clocky is designed to "jump" off the nightstand and run around. The user has the option to set a snooze, but after 1 snooze they must find the robot to shut it off.

I will not implement the jumping functionality at this time, but instead focus on the specifics of having the robot navigate the home environment to avoid the user. I feel that a reactive control architecture is best for this project so that the robot can move and react quickly. Planning movement is not really necessary for this robot to function effectively, so the robot will not benefit greatly from the implementation of a delibritive, hybrid, or BBC system.

The way I imagine the robot working at this beginning stage is to have the alarm set for a particular time to go off. Once the alarm is activated, the robot should immediately head off, hopefully away from the user trying to deactivate it. The robot should be able to avoid collisions with walls, or at least not get stuck on them. Also, the robot should hopefully be able to leave the room, which many Clocky users state that it does not.

Once the user finds the robot, the should be able to push a switch to turn off the alarm clock. If I have time, I will also implement the snooze functionality, but my focus will be on the navigation portion, as I feel this is the key to the success of this particular robot. However, I have some ideas about how to improve this area as well. For instance, the button to turn off the clock can be located on a hard to reach place on the robot, so that the user will need to actually capture the robot to be able to shut of the alarm. The bottom of the robot seems like a good place, but it depends on the exact design of the robot. Another idea is to have some sort of deterrent to being able to capture the robot. The robot can actively move away from the user, or have some kind of action when it's picked up to help avoid being turned off.

I will probably not delve too deeply into the specifics of different alarm sounds or clock functionality at this point. As mentioned previously, I will focus on robot navigation. If I have time these features can be added later.

Project Parts

So I'm about to head to the electronic and mechanical component market here in Seoul, and I need to plan which parts I will need to look for. I think I can build this project with a number of basic components. Here's a list of what I think I will need.

- A chassis to hold all the parts
- A battery pack to power the robot
- Some type of movement effector such as wheels or treads, depending on what's available
- Ultrasonic sensors for wall and object detection
- IR sensors to detect the user, possibly also for wall and object detection
- An LCD display to display the time (included with the SIK)
- Switches/buttons to turn off the alarm, possibly also to use as contact switches (4 included with the SIK, may need more)
- A device to play the alarm sound (the piezo buzzer included with the SIK should do)

That should cover the basics of the system. However, I had an idea to try and deter the user from picking up the device to shut it off by using a spinning device. For this I will also need a motor (included with the SIK), and a some sort of stick, like a popsicle stick or chopstick, which I can find anywhere pretty easily.

Other components that I may need include an additional breadboard to wire the components, but hopefully there are enough jumper wires in the SIK to connect everything. I may also need a new integrated circuit to expand the number of control ports, since I broke mine while going through the SIK guide.

I will head out tomorrow and see what I can find. It may be very hard to find components since I live in Korea and don't speak Korean well (or at all really). If I am unsuccessful, I will have to turn to some online vendors and hopefully be able to ship the parts quickly.

UPDATE

I found a shop that sells Arduino parts and it's fairly near my house. I bought a bunch of cool stuff and met a very nice man who makes some cool Arduino stuff. I also found a bunch of shops where I can order online. Here's a video of all the additional parts I purchased:

https://youtu.be/GxHpvkAMlKg

Project Design

Physical Layout

With all the parts I have, the physical layout of the robot should be relatively easy to set up. I will set my bigger breadboard along the long axis of the chassis. The ultrasonic sensors will be located at the front of the robot, so as to be able to detect objects as it drives around. I will place the IR sensor in the back to be able to detect motion behind the vehicle. When the robot is not moving, if the user attempts to get the robot from behind, the motion will be detected and the robot can react by moving forward. If the user attempts to get the robot from another angle, the ultrasonic sensors will detect this. The other components will be placed onto the breadboard as they fit, and I will try to keep the alarm/clock controls close together since they are all related. The actual RedBoard itself will be mounted toward the rear of the robot, or wherever I can find room for it.

There are a lot of other components to place as well. There is a AA battery pack for powering the whole thing. I will attempt to place this underneath the chassis, but with the motors mounted under there, it may be difficult to find room. The motors themselves are already in place, as the chassis came assembled from the shop that I purchased it from. The RedBoard itself will probably be the most difficult piece to find room for. I can try and attach it underneath also, but I doubt it will fit, and it will be difficult to manage the wiring that way. I may just end up taping it to the battery pack and having placing it on top of the unit, either on it's own or covering up non-critical components.

In regards to the actual attachment of components, The breadboard has a sticky back which I can use to hold it in place. I also want the front ultrasonic sensor to be facing forward, which means that it cannot be plugged directly into the breadboard, so I will probably have to secure it in place with some glue or tape. The same goes for the rear-facing IR sensor. The other ultrasonic sensors might plug into the breadboard directly because they will be used for wall detection along the sides of the robot. All other components should also be able to plug directly into the breadboard. I am not too concerned about wire management at this time, but if it becomes a problem I may have to route some underneath the chassis or tie them off with a twist-tie or something.

Programming Code Structure

I feel that I can divide the features of the robot into to basic functional categories. The first is the alarm/clock functions, the second is the movement behaviour.

Timekeeping Code

For the clock, I will use the RTC unit to sync the system and keep time when the robot is switched off. I feel this unit draws a lot of power so I don't want the display to be constantly on. I will need to figure out how to adjust the power to the back-light or the unit itself so that it is not always on display. The LCD should be on when the alarm begins ringing, and when the user interacts with the device. I will attempt to control this with a button switch and possibly a relay. Other buttons will be used to set the alarm function. At this time, I do not plan to implement a clock setting functionality, because that is not the focus of this project. I will add it later if I have time and can do it easily.

There are also some other issues to consider for an alarm clock. I could implement a snooze button that is standard with most alarms. However, the purpose of this robot is to get you up, and I feel that a snooze function just lets you go back to sleep, so I don't think the implementation of that function is worthwhile. I have also thought about using a puzzle lock to be able to shut off the alarm. This would involve the input of a specific button sequence to be able to shut off the alarm, further causing the user to engage their brain in a wakeful manner. However, this is not a key feature, so I will only add it if I have some time at the end of my project. Finally, there is also the consideration of the time format. I

am going to stick with 24-hour time for now, because it is easier for me to work with and display. Switching to 12-hour time should be fairly trivial and like the combination lock, I will only implement it if I feel I have time and/or it is necessary.

Here is some pseudocode for the timekeeping function of my robot:

```
import libraries
declare constants/variables

setup {

   sync time with RTC
   set up button controls
}

loop {

   if current time == alarm time
      sound alarm

   if alarm on && button pushed
      turn off alarm

   if button pushed
      turn on LCD display
      display current time

   if alarm set button pushed
      display current alarm setting
      move cursor to alarm hour

      if user pushes change button
         change hour

      if user pushes set button
         save hour
      move cursor to minutes

      if user pushes change button
         change minutes

      if user pushes set button
         combine minutes and hours
         save alarm time

   if no button pushed for 30 seconds
      turn off LCD display
}
```

Movement Code

As I mentioned in my earlier project discussions, the robot will implement a reactive control architecture because it is

operating on a short time scale. The user should be able to catch and turn off the robot within a few minutes, and during the time the alarm is not on, the robot remains at rest. The robot also does not have time to plan actions, as it needs to be very quick to get away from the user. The goal is to get far away so the user has to get up and move to turn off the alarm, hopefully putting them in a wakeful state to keep them from sleeping in.

The basic function is for the robot to move while the alarm is on, without running into any objects. I will have the robot move for a certain time, say 10-30 seconds, then pause and wait for the user. If movement is detected, the robot will move away from the detected movement. It will continue this behaviour until the alarm is turned off.

I am using a similar reactive controller as presented in our textbook in Chapter 14 - Don't Think, React! I will need to specify a safe and danger zone for the robot so that it can avoid obstacles. Right now, I am going to implement basic avoidance behaviour, but I may also add some randomness to the movement later to make the robot more unpredictable. Basically, I would use a timer to decide when to make a random turn.

Here is some sample pseudocode:

```
include libraries
declare constants/variables

setup {

   set up pins for motor control
   set up pins for sensor input
}

loop {

   if alarm is on
      move forward for set time

   if front sensor object detection

      if left object detection
         turn right
         continue forward

      if right object detection
         turn left
         continue forward

   else
      turn in another direction (left or right)
      continue forward

   if time over
      stop movement for set time

   if stop time over
      move forward for set time

   if stopped && movement detected
      turn away from movement
      move forward for set time
```

```
}
```

That's it! Time to build the robot and implement the code.

Project Implementation

I've finally coded both the alarm module and the movement module. It took a lot more work than I had initially anticipated.

Alarm Clock

The alarm clock uses the LCD as the main display. The program begins and loads the time from the RTC module. It displays the date on the LCD, followed by the current time and then the alarm time. The display will turn off after 7 seconds if the user does not push a button. If a button is pushed, the timer resets. The user can set the alarm by pushing the set (blue) button while the display is on. This enters the alarm setting mode where the user is prompted to enter in the hours and minutes separately, using the up/down (green/red) buttons, and confirming with the set (blue) button. The hours and minutes are saved to the RedBoard's EEPROM memory, and the alarm is set. If the current time is greater than the alarm time, the alarm is set for the specified time the next day.

The alarm goes off at the specified time, causing the buzzer to go off. It beeps by pulsing once per second and continues until the user pushes any button to stop it. This will also reset the alarm for the same time the next day. Currently there is no way to have the alarm not go off at all, but this is not necessary for the time being. I have not implemented the combination lock functionality to turn off the alarm as of yet, and the time is currently only available in 24-hour format.

The reason this part of the project has caused so much headache for me is because of the LCD. I first set it up as outlined in the Inventor's Kit guidebook. That was easy enough, but it requires the use of many of the RedBoards pins. I wouldn't be able to implement the sensors and motors because there wouldn't be enough slots left. Therefore, I decided to use the 74HC595 IC chip included with the SIK. Holy guacamole it was difficult to do! There are a bunch of different libraries online, the wiring was pretty hairy to figure out, and I also broke my IC (which actually turned out to not be a big deal - see my post)! I eventually figured it all out, so now the LCD only takes up 3 of the RedBoard pins. However, with the current libraries I'm using, I don't have access to the unused output pins available with the chip. I will have to try and figure out what I can do to further reduce my pin usage, because I can't seem to find any shop that sells more of the 74HC595 without buying less than 30 at once. Anyway, for now my alarm clock is up and running. Here is a video of it in action:

YOUTUBE

Sensor/Motor Control

I have the 3 ultra-sound sensors hooked up and my motors are hooked into the RedBoard through the L293D chip. It has been a much easier process to set up compared to the LCD troubles I had earlier. I also discovered a useful tip to get some extra pin space on the RedBoard - you can use the analog pins just like digital ones (I also made a post about it)! I believe this should give me enough space to get all the components in without any more components.

The code I have simulates the alarm turning on after 30 seconds. The robot begins moving forward. Each loop of the program reads from the sensors and stores the values in an array. If it detects it is too near a wall, it will turn away. After 10 seconds, the robot stops moving and waits. When motion is detected, it begins moving again in a direction away from where the motion was detected. These actions continue in perpetuity because there is no way to shut off the alarm for this portion of the program.

The motor control is fairly easy. I move forward by outputting HIGH signals to one of the wires of the motors, with

LOW to the others, just like in the Inventor's Kit guidebook example. Backing up is done by reversing the outputs. Turning left or right is also simple. One wheel moves forward while the other reverses. The only difficulty is finding the timing to be able to turn an appropriate amount. I just calibrated this by running a few trials.

Overall, I'm pretty pleased with the behaviour, but the robot moves a bit too slow for my liking. This is because the motors included in the chassis are pretty small. I would have liked to have at least used the ones included in the Inventor's Kit, but unmounting the small ones and replacing them is not worth all the effort I would have to put in. I'd rather keep the current configuration and just show the robot as a proof of concept, rather than an actual 100% working model. Here is a video of the action:

YOUTUBE

So now, all that remains is to combine these two modules into a final product. Hopefully the process will be smooth. I will make the code available in combined form with the final integration video.

Project Results I - Integration and Testing

Physical Comp

As I mentioned in the previous section on implementation, I was working on combining the alarm module with the sensor/motor module. The first step was the combination of the physical components. This was an easy process, because I had all the circuits planned out with the previous modules. The only concern was to make sure there was enough room on the breadboard. There ended up being plenty of room, because I purchased a large size breadboard for this project instead of using the one that came with the kit. There was a lot of wiring to do, but since I had done it all before, it ended up being quite easy.

In terms of placement for physical components, I did manage to squeeze everything onto my chassis. I held everything in place with electrical tape so that I could easily reconfigure if needed. I put the battery pack at the front, with 3 ultrasonic sensors attached for the front, left, and right directions. The breadboard fit behind the battery pack, along the length of the chassis, and was actually pretty secure, because of 2 little plastic tabs on the chassis that kept it from moving laterally. On the back of the breadboard, I taped the IR sensor into place. There was also a mess of wires to deal with. Extra slack from the ultrasonic sensors was taped to the bottom of the chassis. Most of the wiring on top was left in place as it didn't interfere with the operation of the robot. Finally, the RedBoard I simply placed it on top of the battery pack. I did not secure it with tape for two reasons. First, I wanted to be able to replace the batteries easily. Second, I wanted to be able to access both the DC input and USB inputs of the RedBoard easily as well.

Here are some pictures of the monster that I have created:

Front

Top

Back

Bottom

With everything reasonably in place, I moved on to integrating the code modules as well.

Coding

The integration was very straightforward, the only real work I did was just cleaning up the code for readability and understanding. I had previously designed both modules for just this scenario so I was happy that everything turned out well when it came time to create the full program.

The one problem I did have was not with the code exactly, but it required some minor fiddling with it. I miscounted the number of pins that I was using, and it ends up that I have one too many pins in use! I tried to compensate for this by using the extra pins on my IC. Unfortunately, with the LCD library I'm using in conjunction with it, the extra pins don't seem to be available for use (or at least I couldn't figure out how to do it). However, I did come up with a crude

workaround. The RTC module (which keeps the time when no power is connected) is only used once, in the setup() portion of the code, to read the time and store it in the system. From that point on, the system itself is used for all time calculations and functions. Accordingly, after the system starts up, the pins for the RTC are no longer needed. So I simply unplug one after the initial start up, and plug in the extra component I need, which in this case is the pin for the IR sensor. Of course I have to remember to switch them every time I start up the system, but it is very easy to do and it never caused me any problems.

Here is the fully integrated code if you would like to examine it:

RoboAlarm_Final.ino

*Note - this code uses several libraries that are not included with the standard Arduino IDE. If you have any questions about them, please feel free to contact me.

Expected Performance

So I finally came to the bread-and-butter of the project. Time to put the robot to work. I expect the clock functions to behave as normal, being able to read the clock and set the alarm, etc. I expect the robot to avoid common obstacles such as walls and furniture. If the robot becomes stuck, I expect it to back out of it's position and begin moving again. The robot should move every thirty seconds, and pause for 10 seconds between each movement. If movement is detected during the pause time, the robot will attempt to move away from ths movement.

Actual Performance

I made a video explaining the robot and some of the action.

YOUTUBE

As you can see, the robot behaves reasonably well in a variety of scenarios. However, there were several issues. My original intention was to improve upon the design of the Clocky robot. Overall, I don't think I have achieved this result.

Results II - Discussion and Analysis

So the robot didn't turn out to be 100% perfect. This should be expected of course, but I was still a little disappointed in the results. Here are some of the issues I encountered:

1. Environment

My current home is a loft. As such, I don't have a proper bedroom to test the device. I took the robot into the hallway for filming, providing obstacles with the walls of the room and some cardboard boxes. The robot behaved mostly as expected, avoiding obstacles, and reacting to movement. I tested it in my living room as well, and it was able to avoid walls and furniture there as well, but it did get stuck under my couch a few times because it was the perfect height to be undetectable to the sensors, but still managed to catch on the wires that protrude above the device.

Another issue is that of carpeting. I live in Korea, where most floors are not covered by carpet. This is because the heating system used here is through heated pipes or electric mats under the floor (called 'ondol' heating). To try and simulate carpeting somewhat, I did have to robot move small amounts on my bed and couch. However, the limited space restricted full testing. The robot was able to move in this space, so I would assume that it could move easily on low-pile carpeting without issue.

I also ran tests with piles of clothes on the floor, trying to simulate a messy bedroom environment. The robot was intermittent in its ability to detect this. Of course, it could only detect items that were high enough to be in sensor range. This caused the robot to drag some of the smaller items around with it as it moved, also causing it to stop in some cases because it does not employ very powerful motors (more on that below). Clothing that did meet the height requirement were still somewhat problematic. The robot would run into them at times and get stuck, although most times it was able to back away and get itself out of the situation. I am assuming that it could not detect the clothes because the irregular pattern of the clothing scattered the ultrasonic waves, or the material was able to absorb them somehow.

2. Sensors

I also had other issues with the ultrasonic sensors being unreliable in general. The arrangement that I have, has one forward facing sensor, with 2 more facing left and right at almost 90 degree angles. This arrangement worked quite well for walls and large objects. However, thin objects, such as chair legs, did pose some difficulty. The robot would run into these fairly often. Obviously, one way to alleviate the problem would be to have more sensors to provide a fuller coverage area. However, the bigger issue that I discovered through testing is that my sensors are just not very good. Or at the very least, are not very good for their intended purpose.

I purchased very cheap sensors. They only cost about $3 a piece. When I first got them, I tested them to make sure that they were working properly and calibrated. In static tests, they detected distance quite well and provided quite accurate readings. The next phase of testing was mounting them to the chassis and using them for their intended purpose, but not in coordination with the motors. This proved to work well also. Overall, in both static tests, I was quite happy with the performance. The results were accurate and there was very little, if any, fluctuation in the readings.

The third phase of testing had the sensor inputs controlling the motor outputs. I had the robot propped up so that he wheels could turn without actually moving the robot. When the sensors read an object within a danger zone of 15cm, it would take action by turning or reversing direction, at all other times it would move forward. I noticed that this caused a small but significant change in the behaviour of the sensors. Sensor readings started to become inaccurate. Usually this was on the order of a few centimeters, but in some cases spiked over 10cm. There was a more significant impact on the fluctuation of the readings. The readings seemed to fluctuate around 75% of the time! Yet again, these fluctuations

were usually on the order of only a few centimeters. However, noticing this, I took steps to correct these errors.

Originally, I used only direct readings from the sensors. Each loop of the program ran through the each of the sensors. I had allowed time between each reading to account for any interference from reading one after the other. To account for the fluctuations in readings, I used an array to track the history of previous sensor recordings. If results of successive readings were both within the danger zone, I assumed the readings to be valid enough to have the robot take action. To handle fluctuation, I compared current readings within a certain threshold of previous readings to help eliminate aberrations.

With this solution in place, I moved on the final testing phase. This is where the robot has free movement to act. I noticed a larger change in both the accuracy and fluctuation of the sensor readings. I have no exact data, but the results ended up being fairly unreliable. I kept a similar solution, but tweaked the thresholds that I was willing to accept through trial-and-error. This ended up working for most cases. Nevertheless, the robot would still detect movement when there was none. This I deemed acceptable because the robot is moving most of the time anyway, and I'd rather have it be jumpy than not sensitive enough. The results were less effective for object detection while steering. I found that the robot occasionally would run into walls because they would be undetected. I then instituted a 'stuck' scenario that would compare results of previous readings. If the results were that robot had similar readings after several rounds, it would back out of it's current position and turn around. This worked well because the robot was not moving, providing more reliable results to test if it was stuck.

3. Motors

When I purchased the parts for this project. I found a preassembled chassis with motors already in place. These motors were smaller than those included with the SIK. However, the mounts for the motors only accommodated the smaller size, so switching them out would have been worth the effort. As noted in my previous video experimenting with the SIK motors, there was a torque issue with the motors at low speeds. I was not worried about these issues because the robot would be basically moving at full speed at all times. However, as noted in my project video, I was dissatisfied with the final speed of the robot. My original intention for the robot was for it to be high speed, being able to run away from a person. Ideally this would have been faster than walking speed. Unfortunately, this did not end up being the case. The robot was extremely slow and could easily be caught by anyone. There was no real fix for this other than installing new motors, which would have been difficult, costly, and time-consuming.

Still, this was not the only motor issue I encountered. As the project progressed, a problem began occurring as the robot would not move in a straight line. In the initial testing phases this was not a problem. To get the robot to move forward, I simply set both motors to run at maximum capacity. Each motor running at the same rate produced forward motion in a straight line. When the project progressed further, the movement became less reliable. The left motor began to perform much worse. This caused the robot to turn constantly, instead of moving straight. This behaviour is noticeable in some of my video footage. I tried to correct for this as best I could by having the left motor operate at full capacity while the right was reduced. It took several rounds of trial and error to find the correct amounts. However, this did not completely solve the problem. As the battery power became weaker, the problem would get worse in relation (more on power issues below).

I have several theories as to the culprit causing this problem. First of all, there could have been a load balance issue with the robot, but the turning was significant enough and the components light enough that I believe it could not have been the only issue. Secondly, I may have damaged one of the motors or wheels. I have transported the robot around quite a lot and this could have happened, even though I tried to be very careful. Finally, there could have been debris caught in the axle mechanism. Since the robot runs on the floor, it can easily pick up debris. Hair could have been wrapped up in the mechanism, thus affecting performance. I looked as close as I could and didn't see anything. There could also be issues that I am unaware of such as defective parts and more.

4. Alarm

With the original clock module for this project, when the alarm goes off, a time check is used to cause the buzzer to beep in short pulses instead of producing a long tone. This happens every cycle through the main loop while the alarm is active. On it's own this is fine. With my motor control added to the mix, it caused some issues. To control the motors, I have the program pause for a certain amount of time to allow the steering action to occur. For example, turning left involves having the left motor move at low speeds, while the right moves much faster. I pause the program for a period of time corresponding to how much I want the robot to turn. Yet this causes a problem with the alarm because the main loop executes very quickly, while steering actions take orders of magnitude longer. This caused the alarm sound to be erratic instead of regular. One reason I chose not to change this is because I found this erratic behaviour to be even more annoying! This serves the alarms intended purpose of trying to get someone out of bed to shut it off.

Nevertheless, this implementation would not allow for a custom song or other regular tune to be played as an alarm, which is a feature of many alarm clocks. The best solution for this would be to use threading, having the alarm and steering run on separate threads. I looked into this, but the Arduino hardware does not support threading. The software solutions I found made use of interrupts, and may have sufficed, but only with a significant amount of programming effort. Any tune would have to be cropped into intervals to play, and changes to the alarm playing and motor control functions would be needed. Since the current action sufficed, I chose not to venture down this path.

As small related issue, this lack of threading also had a impact on button responsiveness. The button states were also read once per cycle. This makes turning the alarm off difficult if a very quick button press is used. I almost didn't notice this issue, because in most cases the time to push in a button was not often less than the delay induced by the motor control mechanism.

5. Power

Another issue I had was with power, in two different ways. The first issue was that of power consumption. With the RedBoard, motors, sensors, LCD and more all running, there can be quite a drain on batteries. Because the robot must move independently around a home environment, a cord attached the device is not an option. I used a set of 4 AA batteries to run the robot. I noticed that batteries ran out very quickly. This is partly because I used very cheap batteries, but also because of the amount power used was significant. The second issue is the overall amount of power available. When the alarm goes off, everything is running at once. I noted in my first project video that the LCD would dim slightly when the buzzer was functioning. This problem was exacerbated in the complete model when motors and sensors were added to the mix. The display was not unreadable, but there was a appreciable impact. This was especially noticeable as the batteries began to be quickly drained.

The solution to this is obviously using a larger battery capacity with better output, but I did not look into it. I am not sure how much current and voltage the RedBoard supports, but the output is limited to 5V. Also, most household devices run on AA batteries. I worked within the limitations of the hardware, and just bought a lot of cheap batteries.

Project Future Work

With the lukewarm results of my project, I have come up with several ideas on how my robot could be redesigned for better performance and improved with additional features. This is in addition to the several of the solutions previously discussed.

Improving the Sensors

As mentioned, I encountered several issues with the ultrasonic sensors used to detect obstacles. If I were to start again from scratch, I would probably redesign the robot to eliminate these completely. In an ideal set up, the robot would be moving much faster than mine was able to. As such, even if the sensors were significantly improved, there would still be issues related to unreliable results. A much more reliable and easy to implement solution would be to use contact switches. Each side of the robot could have a switch mounted so that it is pressed when it comes into contact with an obstacle. The robot could then act appropriately to move away from the position of contact. The behaviour may not be as pretty, but it would certainly be capable of being both quick and reliable. An additional benefit is that these buttons could double in function as the actual control buttons for the robot. For example, they could be used to turn off the alarm with a long press, or pressing of all switches at once. All of the other functions occur when the robot does not require the contact switches to sense for movement, so they could also be used to access these functions when the robot is not in the active mode.

Improving the Movement

Improving the motors is something that I discussed in the previous section. Faster/more powerful motors would allow the robot to move at much higher speeds, allowing it to fully achieve its intended purpose of running away from the user. To handle higher speeds, the robot would also need to be more robust. If this robustness is increased enough, it would also allow the robot to fall down stairs or jump off from tables, as I mentioned in my comparison the the Clocky robot.

If the contact sensor idea above were to be implemented, this would allow the robot to quickly and more reliable act on obstacles, thus making it behave better while moving. However, this would limit the ability of the robot to move away from a person attempting to catch it. This could be improved by adding IR sensors to cover the full 360 range around the robot. This would help to move away from a person more easily. I found the IR sensors to be more reliable than the ultrasonic sensors, so better detection would result in better movement away from the user.

Improving the Sound

The alarm right now is a simple buzzer. It is capable of playing simple tones. To make this robot much cooler, a proper speaker could be used. This would allow the user to play actual music and sounds as the alarm. For example, several music or sound files could be loaded into the program and the user could select one to play. This could be further enhanced be having the user upload their own sound/music files so that they can have their own custom sound to wake themselves.

Improving the Clock

The clock right now only displays time in a 24hr format. The user is only able to set the alarm for a specific time within a 24hr period of the current time, and there is no snooze function. The obvious improvement here is to

formats. The standard 12/24hr clocks should be available, but a date function could be added. Currently I have the date display only on startup. Additionally, the user could be able to choose from a number of stylized clock faces, like those currently available on the Pebble or Apple watches. All of these functions would also likely require a better display than the simple LCD used for this project.

Improving the Difficulty (to aid in waking the user)

As previously mentioned, making the robot move faster would make it more difficult to catch. Additionally, improving the robustness of the robot so that it can handle situations like stairs, makes it able to traverse more areas of the home, meaning it can be more difficult to find. Also, implementation of the puzzle combination to shut off the alarm (which I mentioned in the introduction to the project), would increase the difficulty. This could even be enhanced by requiring a mini game of 'Simon' like that seen in the SIK circuit #16.

Improving the Interface

Finally we come to what I think is the coolest idea to enhance this project - adding phone support. I think that the product could be fully integrated with the user's mobile phone through an app. THe alarm and clock functions could be pretty much offloaded from the robot itself, as many people already use their phones as their personal alarms. This would allow the music and sound features mentioned above. It also avoids the problem of the alarm being quieter as the robot moves further from way from the user. The alarm could be shut off via the NFC capabilities in many phones, or some type of short distance connection. After this there could be a requirement for a puzzle lock or some other brain-intensive action to shut off the alarm. This type of integration opens up a plethora of possible interactive features, most of which could be implemented in the app itself, thus leaving the design of the robot itself much more limited in scope. I think this idea is awesome and someone should make it so I can buy it!