Driving Dancing Robots

Anthony Castro, Stephen Martinis, Vashisht Madhavan

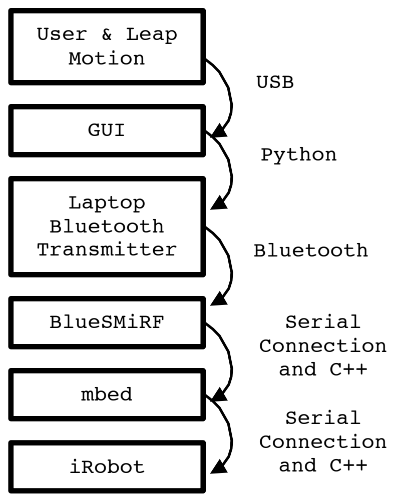
Project Goal: To create a fun, interactive time trial-based game where a user will control a robot based on hand movement.

Equipment:

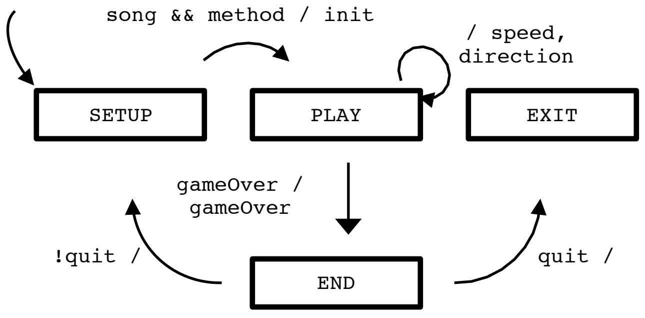
- iRobot Create
- Logic Level Converter
- mbed FRDM-KL25Z
- BlueSMiRF Silver
- Leap Motion
- Laptop

Below are FSM models for the robot. The first is of the GUI and iRobot sub-systems and the second is a synchronous composition

System Model


User & Leap Motion
USB
GUI
Python
Laptop Bluetooth Transmitter
Bluetooth
BlueSMiRF
Serial Connection and C++
mbed
Serial Connection and C++
iRobot

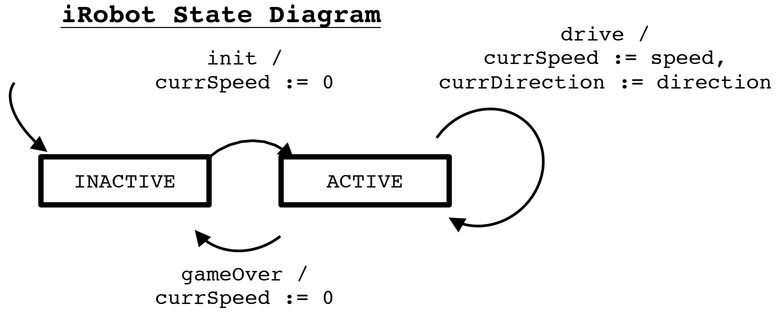
GUI State Diagram



```
Inputs: song: {Fireflies, Head Over Heels, absent}
        method: pure
        gameOver: pure
        quit: pure

Outputs: gameOver: pure   speed: ℤ   init: pure
         direction: {Forward, Backward, absent}
```

iRobot State Diagram

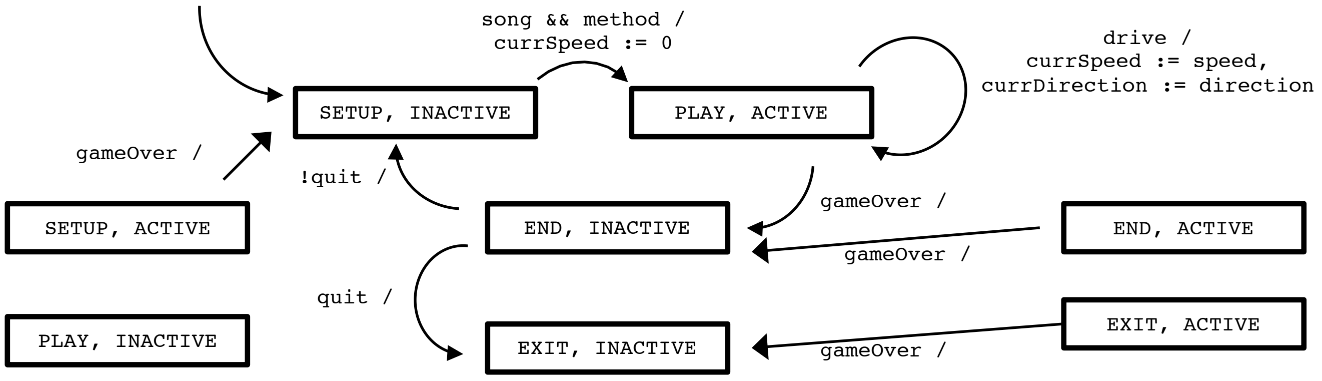

init / currSpeed := 0
drive / currSpeed := speed, currDirection := direction
INACTIVE
ACTIVE
gameOver / currSpeed := 0

```
Inputs: init: pure
        drive: pure
        gameOver: pure
        speed: ℤ
        direction: {Forward, Backward, absent}
        victory: {true, false, absent}

Outputs:        None

Continuous Variables:
        currSpeed: ℤ
        currDirection: {Forward, Backward, absent}
```

Synchronous Composition



```
Inputs: song: {Fireflies, Head Over Heels, absent}
        method: pure
        gameOver: pure
        quit: pure

Continuous Variables:
        currSpeed: ℤ
        currDirection: {Forward, Backward, absent}
```

In the first composition, the iRobot simply reacts to the GUI's outputs. The GUI outputs Bluetooth packets that adhere to the custom Bluetooth protocol. From there, the mbed processes these packets with an algorithm based on the official mbed iRobot tutorial[1] and will output the results to iRobot. We can reduce this composition to form a single synchronous composition as noted. Because of general computation time in the GUI and mbed as well as transmission of data over a wireless unit, this model will have latency and reactions will not be perfectly simultaneous. However, the synchronous composition will accurately model the behavior of the iRobot and GUI. Below are the results of measuring end-to-end latency.

Latency:

The overall average latency of reading a gesture to the iRobot's reaction is given by a summation of the following factors:

- Average Leap Motion latency
- GUI processing time
- Average one way latency over Bluetooth
- ISR execution time
- State diagram execution time

The following methods were used to measure to determine approximate latencies. Procedures were repeated to get an average reading.

- Average Leap Motion latency:
  - Taken from Leap Motion online documentation
  - Approx latency = 30ms
- GUI processing time:
  - Compare timestamp difference when GUI detected hand placement to when a Bluetooth packet was sent.
  - Approx Avg processing time = .0303 ms
- Average one way latency over Bluetooth
  - Get round trip time of packet transmission. GUI sends packet to BlueSMiRF, mbed reads packet from BlueSMiRF via serial connection, sends back to GUI to read. Divide this value by 2.
  - Approx Avg RTT = 48.6 ms
  - Approx Avg one way = 24.3 ms
- ISR execution time:
  - Measure RTT at the beginning of ISR and again at the end of ISR. Calculate the difference between these two readings.
  - Approx Avg ISR exec time = 23.8 ms
- State diagram execution time:
  - Send packet from mbed to GUI before executing state diagram and again after executing. Take the difference between time stamps from GUI reads and subtract 2 times the one-way latency.
  - Approx Avg state diagram execution time = 0 ms
- **Overall appox avg latency = 78.1303 ms**

Since the average latency is approximately 78.1 ms, the composition is not perfectly synchronous, as the iRobot will react to the GUI with a slight delay. Nonetheless, The model accurately shows states that are unreachable and indicate erroneous behavior while also showing proper and accurate behavior of the overall system. For example, the iRobot should never be ready to drive or driving while to GUI is set up as indicated by the unreachable state `(SETUP, ACTIVE)`. The iRobot should also stop driving after the game has ended or the GUI has quit as indicated by the two unreachable states `(END, ACTIVE)` and `(EXIT, ACTIVE)`. We adhered very strictly to these models when implementing the system and during unit and integration testing. By using the models, we were able to locate various bugs in the hardware such as mismatched voltage levels and software bugs such as erroneous packets being sent.

This current model features one-way communication between the GUI and the iRobot. Using mathematical calculations and knowledge of the data packets transmitted to the iRobot, the GUI is able to predict approximately the position of the iRobot, allowing more accurate synchronicity.

[1]http://developer.mbed.org/cookbook/iRobot-Create-Robot

In previous iterations of the system model, we used two-way communication over the Bluetooth channel because of the two conditions for `gameOver`: the song has ended or the goal distance has been traveled. The former case is handled in the GUI, but the latter case initially used the distance calculations in the iRobot Create interface. Since we had a serial connection to the iRobot, we could read the same way we read from the BlueSMiRF, which meant creating an ISR to read from the iRobot. According to the iRobot Create interface documentation, the iRobot sends data packets periodically every 15 ms. Our GUI sends packets according to the period of a quarter note in the song selected (the inverse of the beats per minute). Because data came more frequently from the iRobot than the GUI, the iRobot ISR would constantly execute, preventing the GUI ISR from ever running. We did not want to assign priorities to each ISR, as both were equally important in gathering game information. One potential solution was to implement multithreading where one thread will listen to data coming in from the iRobot and the other will listen to data coming in from the BlueSMiRF. Since they would be accessing shared resources, we would need to implement proper locking. Given the status of our project, time constraints, and perceived difficulty of implementing multiple threads and proper synchronization, we decided not to pursue this route and instead moved towards a different design.

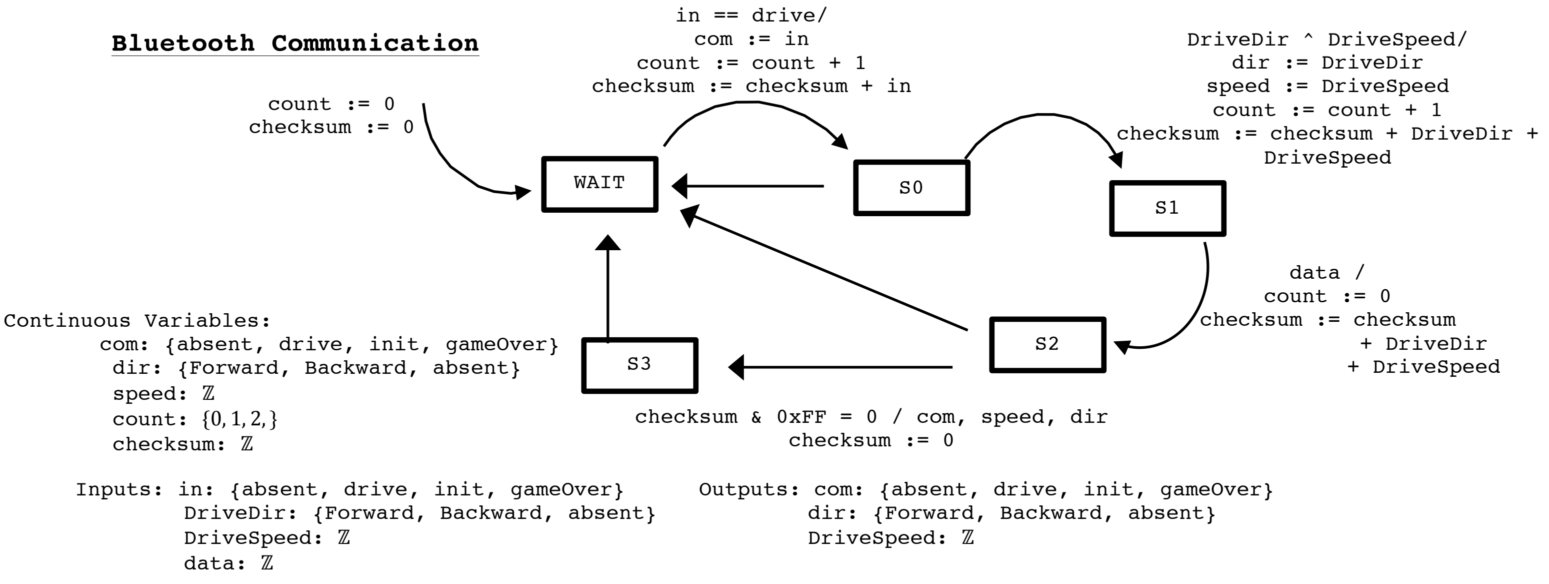
The next iteration featured distance calculations based on the time spent traveling at certain speeds. The mathematical model we used to track distance is as follows:

$$d(t) = \int_0^t v(t)dt = v_0 * (t_1 - t_0) + v_0 * (t_1 - t_0) + \cdots$$

To calculate the distance the robot has travelled, we periodically sampled the velocity, and multiply this by the time since we last sampled the velocity, to get the distance travelled since we last sampled the velocity, in effect integrating the velocity to get the distance travelled.

To prevent further latency, these computations were initially done on the mbed and utilized the mbed timing libraries. However, we discovered that the clock on the iRobot was not functional and did not tick. This led to the final iteration where the distance calculation is in the GUI. Since there would be synchronization issues, we ran several tests in which the iRobot was to travel a fixed distance at several combinations of speed gradients such as a single fixed speed to changing speeds. In each experiment, we found that the margin of error was no greater than 2 cm from the true goal distance. The distance calculation is given below:

Bluetooth Communication:
We created a custom Bluetooth protocol that obeys the following format:
`[OpCode][Packet Data][Checksum]`
In this format, `OpCode` is always mandatory but `Packet Data` and `Checksum` are only necessary when sending driving data. `Checksum` is implemented as a safety measure against data corruption or packet loss. In the either case, the given commands are ignored. The model for decoding the Bluetooth packets is below



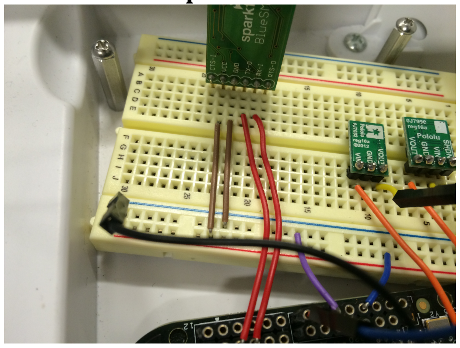
```
Continuous Variables:
       com: {absent, drive, init, gameOver}
        dir: {Forward, Backward, absent}
        speed: ℤ
        count: {0,1,2,}
        checksum: ℤ

Inputs: in: {absent, drive, init, gameOver}
        DriveDir: {Forward, Backward, absent}
        DriveSpeed: ℤ
        data: ℤ

Outputs: com: {absent, drive, init, gameOver}
         dir: {Forward, Backward, absent}
         DriveSpeed: ℤ
```

Hardware Information:

Bluetooth Setup:

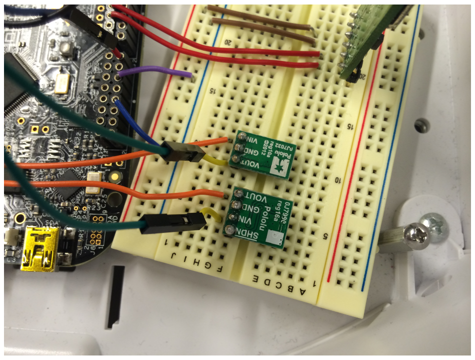


To start off we paired one of our computers to the BlueSMiRF Silver and established an initial connection so that our python program would easily pair every time after. The BlueSMiRF has 5 pins, 4 of which we connected to the mbed via a breadboard. We set the baud rate to be 115,200, the recommended level for the RN-41 chip. We connected VIN and GND pins on the BlueSMiRF to 3.3 VOUT and GND respectively on mbed in order to power the BlueSMiRF. We also connected Tx (transmit) on the BlueSMiRF to Rx (receive) on the mbed to allow serial communication between the mbed and BlueSMiRF.

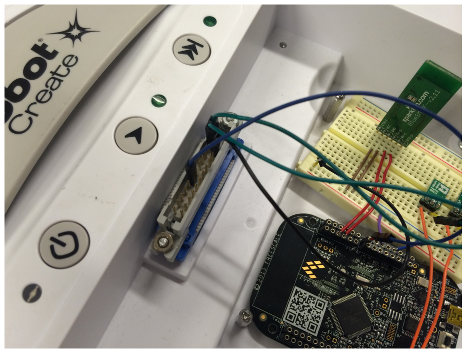
iRobot Connection:

Since there are three UART ports on the FRDM-KL25Z and one was used to buffer Bluetooth data, we use one of the other ports to send data to the iRobot create. We set a baud rate of 57600 for serial communication, the recommended rate for iRobot communication.

One thing to note is that the iRobot transmits data at a 5V level whereas the mbed UART sends data at a 3.3V level. As a result, we needed to add logic level converters between the data connections (Tx and Rx) of the mbed



and the iRobot. We used the iRobot battery to power the mbed at a 5V level. Since the FRDM-KL25Z supports 5V input, there was no need for logical level conversion.



Testing:

We gathered various latency data to get reliable time coordination. This data helped us address some issues about timing coordination. Since we used an ISR to read data from Bluetooth and global variables to temporarily store Bluetooth data, we needed a way to ensure that packets were not getting sent faster than the state machine could process them. In order to do so, we recorded the average time of Bluetooth packet echoing between our computer and mbed and adjusted for the time taken I/0. Taking this account, we adjusted the time intervals between our Bluetooth signals to ensure the iRobot would respond to all data packets recieved.

Conclusion

Overall, we thought this project was a challenging but fun way to explore various technologies while also integrating concepts learned throughout the semester. Using communication latency statistics to effectively coordinate ISR timing and state machine execution, we were able to tackle the issue of concurrency and make it appear as if the iRobot synchronously reacts with hand gestures. In addition, modeling our system as a concurrent state machine simplified the implementation of the driving algorithms helped us identify possible error scenarios. By analyzing the model for unreachable states and, in accounting for the possible ways an error could land our system in one of these states, were able to ensure reliability of the overall system. The use of the Leap Motion kept the project exciting and cutting-edge, adding ‘hacker’ flair to a very structured project.