**Telecommunication and Tele-Controller Subsystem Solution Approach:**

As a company, we see the telecommunication system as the main design objective of this project. For this purrpose, we developed unique communication link for sending commands to a remote location which will become the robot later.

**Communication Link for Sending Commands**

**The Algorithm:**

In this communication link, we utilized a Raspberry Pi 3 as a FM transmitter and used a FM modulated signal which contains command information from the tele-controller unit. Later, at the receiver side for the commands, at the robot, the FM signal is demodulated using a commercial FM radio and using an Arduino Mega 2560, demodulated signal which contains command information from the tele-controller unit is interpreted and respective commands are sent to motors and shooting mechanism. For better understanding, a simplified functional block diagram of the communication link for sending commands is given in figure XX.

We embedded each commands into different sine waves with different frequencies. This way, at the robot, sine wa ves are interpreted as commands and respective functions are executed. First, we have 5 basic commands to operate the robot:

1. Move forward
2. Move backward
3. Turn clockwise
4. Turn counter-clockwise
5. Shoot

For proper operation off full system, we need to overcome noise issue at the FM channel and get a accurate frequency detection and interpretation at the Arduino. The solutions for these issues will be mentioned later in details.



Figure 1 Block diagram of Communication link used for sending commands

**FM Trasmitter (Raspberry Pi 3):**

We used Raspberry Pi 3 (Rpi3) to modulate and transmit a message signal which is a single tone sine wave and used its GPIO4 pin to transmit the FM modulated signal through an antenna. For this purpose, we find a GitHub repository for FM modulation on Rpi3[XX]. However, this program is only created to broadcast a .wav sound file. For our purpose, we needed to tweak the source code of the transmitter program. In our algorithm, we were to send different sine waves for different commands depending on which buttons pressed. First, we implemented a GPIO pin library to program in order to manipulate the program using GPIO pins (WiringPi Library [XX]). Later these GPIO pins are used for detection of which button is pressed. At this point, we decided to embed sine waves as arrays directly into the program instead of creating it in a running programs. This way, the cost of computation time is reduced while the RAM usage is increased. Increase in RAM usage is not a problem because it is very small portion of RAM that is used for this purpose. After creating and embedding the sine waves, we connected the GPIO pin inputs with these sine waves. As a result, when a button is pressed, the assigned sine wave is modulated and the modulated signal is sent to the antenna. A part of modified and tweaked transmitter code can be seen in figures XX. For starter, we only implemented three different commands and therefore three different sine sequences.



Figure 2 Embedded sine sequences



Figure 3 Checking for GPIO inputs and assigning respective sine sequences in main transmission loop



Figure 4 Modulation and broadcast stage of the transmitter in main loop

In figure XX, we declare three different sine sequence. In these arrays, each element represents voltage values for that sine sequence. In the program, depending on the input file, the sampling rate is determined. For the project, we used 44100 kS/s. So, each 1/44100 second a different voltage value is used for modulation. In figure xx, the modulation index is determined such that the bandwidth of the FM signal is 200 kHz which is the bandwidth used for commercial radio receivers.

To choose a useful FM carrier frequency, we searched for empty FM channels between 87.5 MHz and 108 MHz. We found out that at 87.5 MHz, there was no channels and, in our department, the relative noise levels are low. Choosing relatively low frequency is useful because observing the Friis’ transmission equation the free space path loss of the electromagnetic waves is less in lower frequencies.

(n)

After the modulation, the FM signal is to be transmitted via an antenna. We tried different topologies for antennas. First, a simple λ/4 monopole antenna. However, after some field tests, we saw that a monopole antenna could not provide adequate directivity for proper operation. After that we decided to use a commercial TV dipole antenna. After field tests, the results were satisfying, and therefore we stick with the dipole antenna. The physical length was quite long so we may utilize an RF amplifier reduce the antenna gain so the physical length later on the project timeline or we may use higher gain antennas such as dish antenna, Yagi-Uda antenna etc..

Finally, radiating the FM signal through an antenna to the air, the transmitter part is completed.

**FM Receiver (Beats by dr.dre Commercial FM radio):**

After sending commands from the Rpi3 to the air, at the receiver side, the robot, we need to interpret these signals as commands. For this purpose, we decided to use a commercial radio. Because, they have built-in demodulation circuit and automatic gain control circuit (AGC). Also, they have better tuning capability then we can make in such a short period of time as the project timeline. To use a radio in our robot, we cut one of the earphones of the radio and soldered jumper wires to ends. By doing so, we have an electrical signal that have the command information which is a single tone sine wave. This signal will be fed to Arduino Mega 2560 for interpretation at later stage.

**Interpreter (Arduino Mega 2560):**

To use the received and demodulated signal, we need to interpret it as commands. For this purpose we used Arduino Mega 2560. Since our commands are assigned to sine waves with different frequencies, we need to determine the frequency of the received signal. We impleneted a piece of code which is given in figure XX to count the frequency of the input signal. As can be seen in figure XX, we put a threshold value for detection of the signal periods. Also, to minimize the effects of channel noise on detection, we implemented a moving average filter which waits for three samples then averages them. After that, the average period of received signal is calculated. After measuring the period, simply the frequency is obtained. Later, this detection is used for motion subsystem to execute the commands.



Figure 5 Arduino code for detection of the frequency of received signal

To choose proper sine frequencies to be assigned to a certain command, we needed to take into account that how well and with how much error rate can the Arduino can detect the frequency of the received signal. So we performed an experiment to determine the percentage error rates at different frequencies form 50 Hz to 20kHz. The results of the experiment can be seen in figure XX. Observing the figure xx, we can say that at higher frequencies (>6kHz), the error rate is relatively high. And at low frequencies below 6 kHz, especially lower than 2 kHz, the error rates are between 1% and 2%. Although we used 2kHz, 4kHz and 6 kHz sines for the general test and submodule demo, for the final product, we will use different frequencies for the commands regarding the measured error rates.

Figure 6 Maximum error rates of detected frequencies of different received signals