Introduction
The purpose of this project is to let the robotic car we designed to move from a start point to an end point by using 8 Tarokos and 2 infrared sensors. Among 8 Tarokos, one of them is control node (sink node) used to control the car and to receive data from source node, one is a tag represents the location of our car, one is a source node which processes the information it retrieves from position system and transmit the location data or commands to the sink on the car, and five of them are relay nodes to relay data from source to sink. The two infrared sensors are used to detect obstacles or walls in front of the car.

Test Environment
The figure above is the floor plan of 6F of BL building. Given a start point and an end point, the car has to be able to finish the tour without human control. All our resources are listed as follows.
- 8 Tarokos
- 2 Infrared sensors
- 1 printed circuit board (PCB)
- 1 robotic car with 2 step motors
- 2 battery slots which can be inserted with four batteries each

Hardware Design
1. Circuit Board Design
In this semester, we did not use the breadboard. Instead, we used the Altium
Designer tool to design our own circuit and sent the layouts to factory to get a PCB (Printed Circuit Board.)

On the upper-left and upper-right sides are sockets for infrared sensors (I1, I2.) The lower-left and lower-right sides are sockets for left and right motors (M1, M2.) There are two slots for us to install the two Tarokos, which one of them is a control node (sink, T1,) and the other is the tag (T2.) For the control node, we used 2 pins as inputs to receive signals from infrared sensors, and 2 pins as outputs to control the rotation of wheels. In the middle of our PCB, there are a power supply slot (B1) and a LM317 slot (L1), which is used to convert the voltage form 6V to 3.3V. Power source of 6V is for motors and infrared sensors and power source of 3.3V is for Tarokos.

2. Infrared Sensor Placement

We put 2 infrared sensors in the left-front and right-front sides of the car which is shown in the photograph. We let the infrared sensor face 45 degree out from the moving path. If we put both infrared sensors right in front of the car, facing the same direction of car travelling, then they will not be able to distinguish walls on the left- or right-hand side of the car.

Strategies

1. Position Calibration

Due to the low accuracy of raw fingerprint-based positioning, we designed a calibration algorithm which utilizes not only the average but also the standard deviation of a set of given positions to filter out the outliers.
For a given set \( P \), which contains \( n \) position data, the general idea of the algorithm is to compute the average \( u \) and standard deviation \( v \) of the set and eliminate the data that are not in interval \([u-v, u+v]\). We repeat this until there is no datum eliminated and we take the average of this final set as the result. The pseudo-code is as follow.

```plaintext
calibrate (a set of possible positions: \( P \), size of \( P \): \( n \)) {
    int u = avg(P);
    int v = var(P);
    int cnt = 0;
    Set Q = {};
    for each position \( p \) in set \( P \) {
        if \( (p \text{ is in } [u-v, u+v]) \) {
            put \( p \) in \( Q \);
            cnt++;
        }
    }
    if \( (n == cnt) \)
        return u;
    else
        return calibrate(Q, cnt);
}
```

For example, for a set \( P = \{1, 1, 2, 2, 9\} \), clearly 9 is an outlier. If we naively take the average, than the result will be 3; but with our calibration, we can get a more accurate result, 1.5.

Although the complexity seems to be unbearably \( O(n^2) \), but the size of \( n \) is actually quite small in practice (\( n = 16 \) in our case.) Another issue is the feasibility of data elimination: what if there is no datum within the \([u-v, u+v]\) interval, i.e., \( cnt = 0 \)? It can be proved that the situation will never happen. A sketch of the proof is like this: take the worst case, data in the set are evenly divided into two groups, say \( P = \{1, 1, 1, 9, 9, 9, 9\} \). This is the case that all data are far from the average, but relatively, the deviation will grow larger, too, making the interval large enough to contain at least one datum. (Also, this is the case that our calibration can do little good.)

2. Basic Obstacle Avoiding

We tuned the detection distance of the infrared sensor like the table shown below. The NEAR state means that the distance between the obstacle and infrared sensor is
less than 10cm. The MID state and FAR state mean that the distance between the obstacle and infrared sensor are between 10cm and 20cm, and larger than 20cm, respectively.

<table>
<thead>
<tr>
<th>Distance (infrared and obstacle)</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 10cm</td>
<td>NEAR</td>
</tr>
<tr>
<td>10cm &lt; distance &lt; 20cm</td>
<td>MID</td>
</tr>
<tr>
<td>&gt; 20cm</td>
<td>FAR</td>
</tr>
</tbody>
</table>

Take a simple example, if the left infrared sensor is in the MID state and right sensor is in the FAR state, it means that left side of the car may be a wall, and the car needs to turn right. Some basic movement decision of the car is listed in the table shown below. As for decisions for other state combinations, we will discuss more details in the next subsection.

<table>
<thead>
<tr>
<th>Left infrared sensor</th>
<th>Right infrared sensor</th>
<th>movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>FAR</td>
<td>Turn right</td>
</tr>
<tr>
<td>FAR</td>
<td>MID</td>
<td>Turn left</td>
</tr>
<tr>
<td>FAR</td>
<td>FAR</td>
<td>Straight forward</td>
</tr>
</tbody>
</table>

3. Area Coloring
We divided the floor plan of BL-6F into different areas marked by colors.

Base on the direction of movement (clockwise or counter-clockwise,) the car has different moving policies in different color areas.
The reason to distinguish blue and purple areas is to take advantage of the walls. Due to the inaccuracy of position system, our car may not turn as soon as it enters the colored areas. Thus, we utilize the walls in the end of a straight corridor to let the car know that it should turn 90 degrees immediately. However, this method does not work for purple areas if we travel clockwise. If we use the same method in purple areas, our car will enter a red area (dead end) before it meets the ending wall. Therefore, our car is designed to be able to turn automatically without the help of walls when it travels clockwise into a purple area.

Now that we have marked up the critical areas, we can set different behaviors for our car relatively to different infrared states. Notice that in the table below we assume the car is travelling in counter-clockwise direction. As for clockwise travelling, the policies will be opposite.

<table>
<thead>
<tr>
<th>Color</th>
<th>Clockwise</th>
<th>Counter-clockwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>Straight forward</td>
<td>Straight forward</td>
</tr>
<tr>
<td>Red</td>
<td>Back up</td>
<td>Back up</td>
</tr>
<tr>
<td>Blue</td>
<td>Right turn</td>
<td>Left turn</td>
</tr>
<tr>
<td>Purple</td>
<td>Automatic right turn without the help of walls</td>
<td>Same as in blue area</td>
</tr>
<tr>
<td>Yellow</td>
<td>Left turn</td>
<td>Right turn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Color</th>
<th>Left infrared sensor</th>
<th>Right infrared sensor</th>
<th>movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue &amp; Purple</td>
<td>NEAR</td>
<td>FAR</td>
<td>Towards right rear</td>
</tr>
<tr>
<td></td>
<td>FAR</td>
<td>NEAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>MID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEAR</td>
<td>NEAR</td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>NEAR</td>
<td>FAR</td>
<td>Towards left rear</td>
</tr>
<tr>
<td></td>
<td>FAR</td>
<td>NEAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>MID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEAR</td>
<td>NEAR</td>
<td></td>
</tr>
<tr>
<td>White</td>
<td>NEAR</td>
<td>FAR</td>
<td>Towards left rear</td>
</tr>
<tr>
<td></td>
<td>FAR</td>
<td>NEAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>MID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEAR</td>
<td>NEAR</td>
<td></td>
</tr>
</tbody>
</table>
The general idea is to replace right (left) turns by moving towards left (right) rear direction when at least one of the infrared sensors senses a NEAR state. When infrared sensor is in NEAR state, it means that there may not be sufficient space in front of the car for it to do right/left turn. Thus, we let the car to retreat with different turning speed on each wheel and accomplish back off and turning at the same time.

The left rear back off policy in white areas is specifically designed for the white area <12>. As depicted in the figure, the corridor in area <12> is not simply straight. It contains multiple pits which can cause disaster to our car if it naively treats it as normal straight corridors. With the help of left rear turning, if the car goes into a pit and bump into its corner, it will move toward left rear until both infrared sensors are in FAR state. The car will repeat the procedure until it gets back to the normal route.

In addition, purple areas are the special cases in the map. First, when travelling in counter-clockwise direction, due to the fact that every purple area has a red area neighbor on its right, it is dangerous for cars to take right turns in purple areas. Thus, if the car detects that the states of infrared sensors are (L=MID, R=FAR) in a purple area, instead of turning right as in other areas, it goes toward right rear. Second, when travelling in clockwise direction, the car needs to turn right without the help of walls. To accomplish this, we modified the forwarding function to let the two motor to turn in different speeds. As result, the car leans to the right when it goes ‘forward’ in purple areas.

<table>
<thead>
<tr>
<th>Special Movement In Purple Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clockwise</strong></td>
</tr>
<tr>
<td>When the car goes forward, it leans to the right.</td>
</tr>
<tr>
<td><strong>Counter-clockwise</strong></td>
</tr>
<tr>
<td>When the car wants to turn right, it goes right rear instead.</td>
</tr>
</tbody>
</table>

4. U-turn Detection
After continuously avoiding a series of non-wall obstacles like vases, umbrella rack, or waste bins, our car can travel in an opposite direction with bad luck. Thus, a mechanism that can detect and correct the direction will be necessary.
We set a central point of the floor plan as the origin $O$. For every $t$ seconds, our U-detect algorithm takes a sample from the current position $P$ of the car. By applying inverse trigonometric function on vector $\overrightarrow{OP}$, we can determine an angle $\theta$ of the position. For clockwise traveling, $\theta$ should be an increasing variable to time. (Notice that the y axis in the figure points to the bottom.) By detecting the difference between two consecutive sample angles, we can accomplish false direction detection. Pseudo-code is as follows.

```plaintext
Double theta0;
Int Ox, Oy; //the origin
uDetect (Current position of the car: (CURx, CURy)) {
    theta1 = atan2(CURx-Ox, CURy-Oy);
    diff = theta1 - theta0
    theta0 = theta1;
    If (CLOCKWISE && diff < 0 || !CLOCKWISE && diff > 0){
        //wrong direction
        uTurn();
    }
}
```

We set $t = 60$ seconds in practice. This rather lengthy sampling interval is to minimize the probability of false positive due to positioning inaccuracy.

5. Workload Distribution
Except of sink and tag Tarokos, all the other Tarokos are plugged to abundant, steady power supplies, either to a socket on the wall or to a laptop. In addition, the performance of motors and infrared sensors are extremely sensitive to voltage variation. Thus, we bias the computational workload to let the source Taroko do most
of the computing, including color area detection, U-turn detection, and position calibration. Instead of coordinates, our source Taroko only transmits the final results (colors, need U-turn, or GOAL!) of the signal processing previously mentioned so as to alleviate the workload and to minimize the power consumption of Tarokos that rely on batteries.

**Challenges**

1. **Position System Breaks Down**

   During the development of this project, client side of the position system seems to be very unstable. In our worst experience, it only lasted less than 2 minutes before the tags disappeared and never showed up again until you rebooted the client. As a counter measurement, we need to design some fault tolerance for our Tarakos.

   When the position system breaks down, (no matter sever or client side,) it starts to transmit coordinate (0, 0) to our source Taroko. Therefore, as soon as our source Taroko gets (0, 0), it signals the sink Taroko to halt the car and to notify the users by toggling the LEDs. After the users notice the break down and reboot the position system, our source Taroko can seamlessly continue its work.

   But the break down can still slightly influence our performance. When the system break down, the position set of our calibration algorithm will be stuffed with zeros, which can cause some minor effect on the accuracy in the beginning after reboot.

   (At the night before final demo, TA released a new steadier version of the position system client so this fault tolerance design did not make a use in the actual demo.)

2. **Hardware Problems**

   We met some problems when we put all the elements on PCB. First, the voltage for the Tarokos was not exactly 3.3V. We found that it was because the resisters were not welded correctly. Second, the infrared sensors were not connected to the PCB. There was a gap between the socket and the wire. In order to solve this problem, we let the stitch on the socket be thicker by adding tin solder. Third, at first, we just used four batteries which provide 6V to the PCB. When we turned on the radio receiver of Taroko, the movements of the car became unpredictable. We guessed that the voltage provided to the Tarokos is too low. So, we paralleled the power supply with another four batteries, and the movements of the car became normal. We thought that the load was too large that four batteries were not enough, so the voltage dropped dramatically.
3. Color Area Detection
All colored areas except the white ones are critical regions like corners and dead ends where the car not only needs to avoid obstacles but also to turn or to backup automatically. After several field trials, we found that the inaccuracy of position system tend to omit these areas. When the tag travels through a straight corridor, goes around a corner, and enters another corridor, its positions shown in the system are rather in the first corridor in the beginning and just ‘teleported’ to the other corridor. The critical corner area was totally omitted.

Thus, we tried to manipulate our color decision function in order to fit the inaccurate feedbacks of the system (Sort of doing a coarse training phase for the position system all over again.) We got two adjusted version of the map, based on the direction of traveling of the car.

New map for counter-clockwise travelling

New map for clockwise travelling

Generally, critical areas in the new maps are extended by different levels toward the opposite direction of car travelling. The different levels of extension are based on the actual ground measurements. Take the area
in front of the GIPO office, which has the greatest level of extension, the position
system tend to identify it as the area in front of Lab 603. Therefore, we need to
extend the blue area rightwards for clockwise travelling case, in which our car will
come from the right side.

**Reason of Failure**

1. **Demonstration 1: Power dropped down**
   As we previously mentioned, if the voltage of the power supply is too low,
   movements of the car become unpredictable. The reasons may be that the threshold
   of the infrared sensor changes or that the signals generated by control node are
distorted. However, we didn’t expect that the situation happened during the demo
time. We should have exchanged the old batteries with new ones.

2. **Demonstration 2: Human cause**
   Regretfully, when we luckily got a second chance of demonstration, one of our team
   members loaded the wrong version of code onto the sink Taroko. There is a bug in
   packet sequence checking which incurs the sink to ignore every packet after an
   amount of time. Sadly, our car failed to receive any commands sent by the source
   and became totally blind in the end.

**Postscripts**

Chien-nan Chen

As a networking major, I rarely have the chance to program on a micro-controller or
to weld resistors on circuit board. Problems like power sufficiency, baud rate
 tuning...etc. are brand new experiences to me. As for the results of final demo, well,
there is a saying in my previous research topic, security, during undergrad: strength
of a chain depends on its weakest link. In our case, the weakest link would be human
issue. Loading the wrong version of code is killingly unacceptable, especially when
the guy that actually done the debugging and coding and testing is different from the
one that made the mistake.

Po-Hung Lin

In this class, what I learned are not just how to use the infrared sensor and control
the motor wheels. I also learn from the problems when I combined all the hardware
together and from the problems when we tested our car moving around at 6F of BL
building. Even though I thought we’re well-prepared, something wrong happened. I
felt really frustrated and depressed. However, I must thank for my partner. He not
only helps me a lot, but teaches me a lot. At the end of this semester, this course let
me think of a sentence “life is like a box of chocolate, you never know what you’re gonna take.”