Gesture Controlled Driving

Ollie Peng, Manish Raghavan, Victor Sutardja

Overview

Using the Kobuki platform from lab, we built a system by which the Kobuki drives in a path drawn by the user. Specifically, the user draws a path in the air using a brightly colored object. The Kobuki takes a series of images of this path using a webcam mounted on its frame. These images are streamed to a laptop which performs the image-processing needed to detect the object used for drawing. The points from the images are interpolated into a smooth path. Using feedback from the OptiTrack camera system, the Kobuki is instructed to drive along this path, correcting for any errors in real time.

Setup

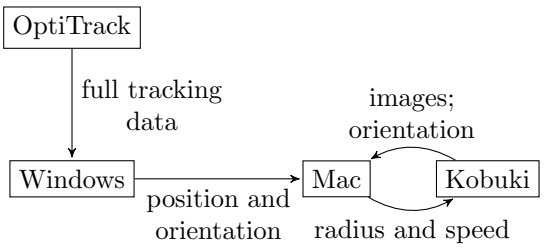

Figure 1: Flow of information

Figure 1 shows the flow of information between various devices. There are two key design decisions we made in order to overcome challenges we faced. First, we streamed the OptiTrack data to a separate Windows machine instead of the Mac that was communicating with the data because the frame rate from the traking system was too high and would be constantly modifying shared variables. Since we were only sending updated commands to the Kobuki once every second, we did not need such frequent information, and therefore used the Windows machine to sample the data at a lower rate before passing it on the Mac. Second, we chose to take orientation data from both the OptiTrack and Kobuki. We did so because the orientation from OptiTrack was given in terms of quaternions in a rotated axis space, meaning that the standard equations for converting to yaw, pitch, and roll no longer applied. Instead, we needed to know which quadrant the yaw was in to correct them. As a result, we used the orientation from the Kobuki to determine which quadrant it was facing, allowing us to use the correct equations to get the true orientation.

Image Processing

Compare to naive and say why ours is better

B-spline Interpolation

We chose B-spline interpolation with cubic polynomials because it has the following properties: [1]

- Locality – only nearest 3 points affect the curve
- Continuity – continuous up to the second derivative

B-spline interpolation yields a smooth path, allowing the Kobuki to drive without stopping to turn. Intuitively, the interpolation yields a path that is at any time a linear combination of three neighboring points, weighted by the basis curves at those points. Since the basis curves, shown in Figure 2 approximate Gaussians when in the middle of the path, the interpolation can be viewed as letting each control point have "influence" on the interpolated curve that depends on how close the curve is to that point. However, this means that the interpolated curve does not actually pass through the control points. As a result, we had to take in a series of data points and find control points such that the resultant curve interpolated through the control points passes through the original data points. Figure 3 shows how from the original data points, shown in blue, a series of control points, shown in red, are computed, and when interpolated, the final path passes through the data points.

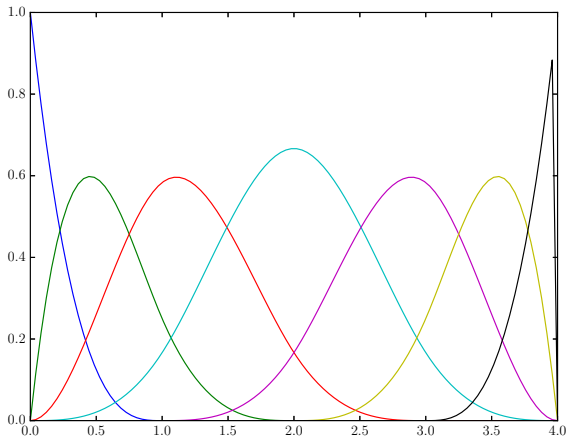

Figure 2: Cubic spline basis curves

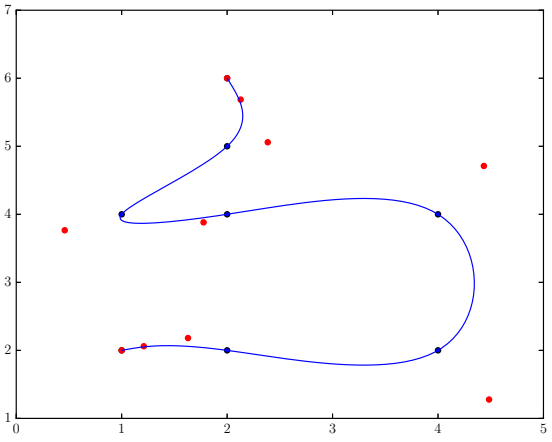

Figure 3: Sample cubic spline interpolation

By convention, we will assume that we have $n+1$ data points. To ensure that the endpoints of the curve are correct, we define

$$t_i = \begin{cases} 0 & 0 \le i \le 3 \\ i-3 & 3 \le i \le n+3 \\ n+3 & n+3 \le i \le n+6 \end{cases}$$

The equations for the basis functions are given by the following recurrence relation: [2]

$$N_{i,1}(x) := \begin{cases} 1 & \text{if } t_i \le x < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(x) := \frac{x - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(x)$$

In order to get the curve to pass through all the data points, we need 2 more control points than data points. To find the control points from the data points, we use the fact that the curve must start and end with our first and last data points and must pass through each of our data points. Since we have $n+3$ control points and $n+1$ data points, we add the initial conditions that the second derivative of the curve at the start and end points must be 0. This yields the following system of equations:

$$\begin{aligned} \mathbf{P}_0 &= \mathbf{D}_0 \\ \frac{3}{2}\mathbf{P}_1 - \frac{1}{2}\mathbf{P}_2 &= \mathbf{D}_0 \\ N_{i,4}(i)\mathbf{P}_i + N_{i+1,4}(i)\mathbf{P}_{i+1} + N_{i+2,4}(i)\mathbf{P}_{i+2} &= \mathbf{D}_i \\ -\frac{1}{2}\mathbf{P}_n - \frac{3}{2}\mathbf{P}_{n+1} &= \mathbf{D}_n \\ \mathbf{P}_{n+2} &= \mathbf{D}_n \end{aligned}$$

Solving for these gives us the control points $\mathbf{P}_0, \ldots, \mathbf{P}_{n+2}$. To interpolate the curve, we simply use

$$C(t) = \sum_{i=0}^{n+2} N_{i,4}(t)\mathbf{P}_i$$

Timing

- Frame rate from OptiTrack
- Mac polling Kobuki
- Kobuki polling Mac
- Threads
- Mac polling Windows and sending to Kobuki

Threading

In controlling the Kobuki, we had to send it instructions while also receiving orientation data from it. To do so, we used threading – one thread listened for data from the Kobuki while the other thread used that data to control it. In order to prevent race conditions, we used locks on two shared resources: the socket over which we were communicating and a global variable containing the Kobuki's orientation. Since we had two threads each using the same two locks, we had to avoid deadlock, which we did by ensuring that no thread held more than one lock at the same time. Each thread executed a task periodically, on the order of tenths of a second or longer, meaning that the delay caused by waiting for locks was insignificant compared to the running time allowed for each task.

Modeling Movement and Control

The Kobuki requires a radius of curvature and wheel speed to control its motion. Our control algorithm takes in the Kobuki's current position, the next position it should go to, its current orientation to produce these quantities. Let $\mathbf{p}_1 = (x_1, y_1)$ be the Kobuki's current position and $\mathbf{p}_2 = (x_2, y_2)$ be the Kobuki's next position. Let $\mathbf{p}_\Delta = \mathbf{p}_2 - \mathbf{p}_1$. Let ϕ be its current orientation, measured counterclockwise from the positive x axis. The angle α between its current position and its next position is given by

$$\alpha = \text{sign}(y_\Delta) \cos^{-1}\left(\frac{x_\Delta}{||\mathbf{p}_\Delta||}\right)$$

The total angle it must turn is $\theta = \alpha - \phi$. The radius of curvature is then given by $r = \frac{||\mathbf{p}_\Delta||}{2\sin\theta}$, which is positive if

turning left and negative otherwise. The angle through which it must follow an arc of this circle to reach $\mathbf{p}_2$ is 2θ, so the speed at which it must travel is $s = \frac{2r\theta}{\tau}$, where τ is the time alotted for it to reach $\mathbf{p}_2$. Figure 4 shows the model of the Kobuki's path from $\mathbf{p}_1$ to $\mathbf{p}_2$.

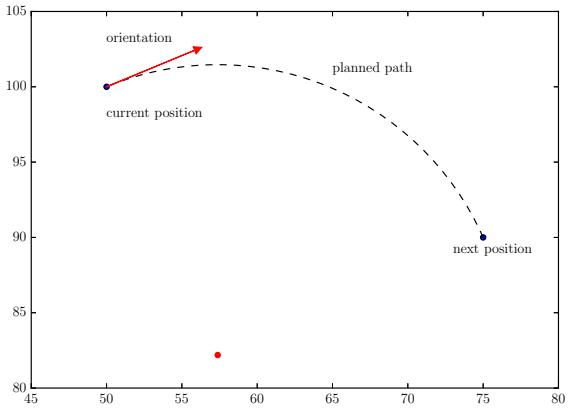

Figure 4: Movement planning

Challenges

One of the major issues we faced was communication between devices over networks we could not control. Because AirBears2 blocks traffic on some ports, we had to set up our own local networks, while remaining connected to the Internet to stream OptiTrack data. In such a complex network, new problems arose arbitrarily as factors outside our control made our communications unpredictable.

Results

Shown in Figures 5 and 6 are examples of the Kobuki's actual path (shown in blue) compared to the planned path (shown in red), as tracked by the OptiTrack system.

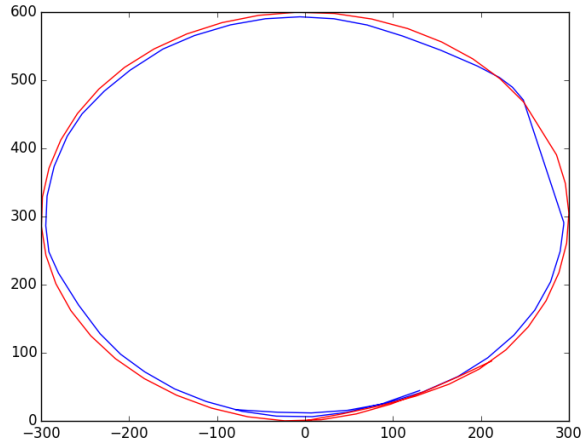

Figure 5: Actual vs. planned path with low third derivative

For the path shown in Figure 5, the RMS deviation from the path is 6.074 mm. The deviation occurs because the Kobuki is always in a sense "catching up" to where it should be on the path, so it tends to cut the corners.

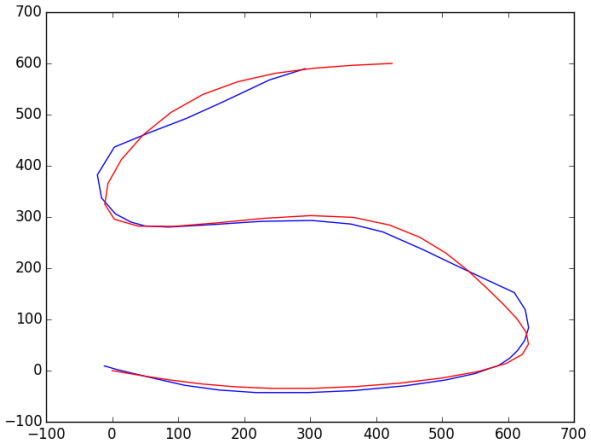

Figure 6: Actual vs. planned path with high third derivative

A video of the Kobuki following the path shown in Figure 6 can be found at `http://www.youtube.com`.

Extensions

In order to improve performance on instances with high third derivatives, we could either send commands to the Kobuki more frequently or reduce the speed at which we expect the Kobuki to follow the path. Doing either would allow us to react to deviations from the true path more quickly, thus reducing the overall error.

Since our final control algorithm only used position, next position, and orientation, it could be easily extended to allow the Kobuki to follow a moving object. As long as the Kobuki's position and orientation are sampled frequently enough, it could follow a moving object using the same techniques.

References

[1] Fuhua Cheng. Curve interpolation using uniform cubic b-spline curves. http://www.cs.uky.edu/~cheng/cs631/Notes/CS631-Chap3-3.pdf, Sep 2012. [Online; accessed 12-December-2015].

[2] Wikipedia. B-spline — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/B-spline, 2015. [Online; accessed 12-December-2015].