Controlling the iRobot Create Platform via Overhead Projection of the Scene

Rasid Pac

December 14, 2011

Introduction

iRobot has created the iRobot Create, providing researchers and hobbyists alike with a versatile and easy to use platform based on their robotic vacuum cleaner, the Roomba. While there are a multitude of ways of designing a control system for this platform, we have succeeded in creating a point to point and trajectory tracking system for the iRobot Create platform by utilizing color detection from a camera placed in an overhead view. By enabling the user to simply click on a location in the image plane provided by the video feed, we have developed an easy to use and intuitive system for controlling a Create robot.

Robot Interface

Communication with the iRobot Create is accomplished with iRobot Create Open Interface which utilizes two way, serial communication at TTL levels (0 – 5 volts). For the requirements of this project, data is not retrieved from the Create, only sent. Data is transmitted in eight bit opcodes, with each having various functionality. The hardware connection is made with the robot via a seven pin mini-DIN connector. The other side of the cable connects to the controller, in this case a personal computer, with a serial port following the RS-232 standard. Controlling the Create proved to be one of the most difficult challenges of this project due to the nature of the Windows 7 platform and Windows Visual Studio. At the time of this writing, there does not appear to be a well-documented Windows-based communication library for the Create with the most popular open source libraries being written for Linux distributions. Due to a lack of compatible software libraries, we set out to create our own class, Creabot, that would handle the robot interface. To do this, we started with a serial communications port class which was modified for compatibility with Open Interface. The heart of the Creabot class is the directDrive function which receives the output of the control algorithms: the translational velocity and the angular velocity. Since the model for the iRobot Create is a unicycle robot, it has a built in function for the direct drive opcode which allows the controller to set each wheel speed independently. Exploiting this extra degree of freedom provided a high level of control of the robot's trajectory by being able to quickly compensate for and react to large variations in the values of the translational, but more so angular velocities.

Robot Localization

Determining the location of the robot based on camera feedback is accomplished using a color detection algorithm developed during the course of the project. Two colors, red and blue, are used to find both the position and orientation of the robot. Circular patches of red and blue papers are mounted on a black cardboard with precise alignment and symmetric positioning as shown in Figure 1. The cardboard is then attached to the robot with the red blob facing the front of the robot. The center of mass of the cardboard is aligned with that of the robot.

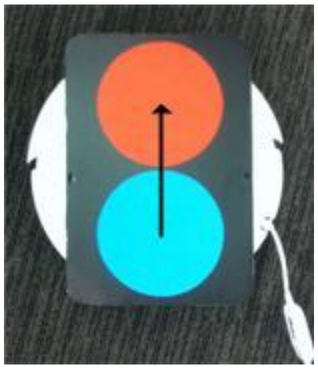


Figure 1 Robot pose determination using two color blobs

The robot localization algorithm detects the two color blobs on the robot and finds the center of mass of each blob. Then, the vector between these two centers provides the orientation information for the robot. The position of the robot is then simply the average of the two center points as shown in eq. (1).

$$x_r = \frac{x_{red} + x_{blue}}{2};\ y_r = \frac{y_{red} + y_{blue}}{2};\ \theta_r = \operatorname{atan2}(y_{red} - y_{blue}, x_{red} - x_{blue}) \tag{1}$$

Color Detection

Simple detection of the red and blue colors attached to the robot requires having a background without any those colors in it. ASTRA lab carpet floor is mostly a mix of black and gray colors so it is adequate for using red and blue blobs for detection.

In order to distinguish a certain color in an image, the hue and saturation information of the pixels are used as distinguishing colors in HSV space can be done by eliminating the effect of illumination intensity. Hence, the image captured from the camera is converted from RGB (Red-Green-Blue) space to HSV (Hue-Saturation-Value) space. After this conversion, the image pixels are passed through a threshold filter such that only those pixels with matching H and S values are selected to form a monochromatic image with corresponding white pixels. The following figure shows the distribution of colors in the HSV space. It can be seen in part (b) of the figure that elimination of the V component does not affect the color information. Also, red and blue are about 120 degrees apart on the H plane hence are easy to distinguish.

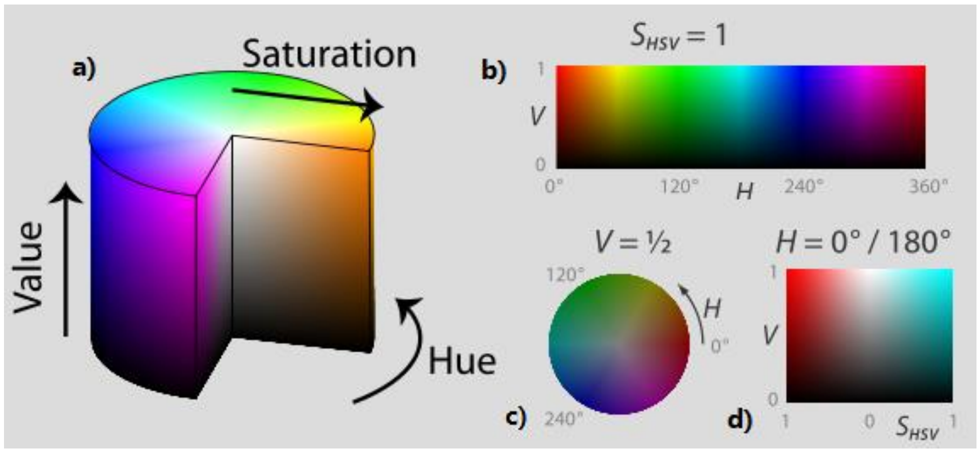


Figure 2 Hue, saturation, and value space [2]

The thresholds for red and blue color hue and saturation intervals are found based on images taken from the scene. The user selects a red blob region and the corresponding lower and upper thresholds are set by adding and subtracting a constant value to the average hue and saturation. The same procedure is repeated for the blue color as shown in Figure 3.

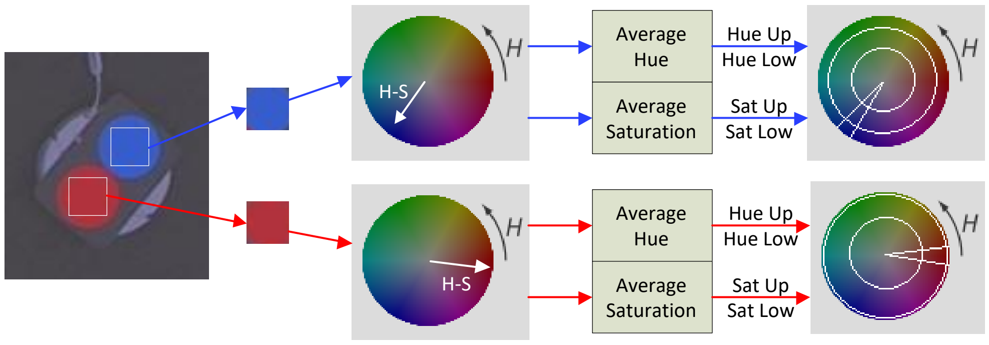


Figure 3 Extraction of upper and lower thresholds for hue and saturation values

-mathematical relationship between blob position/orientation and trajectory
-image parameters such as threshold
-drawing trajectory, plotting path taken
-discuss user input details
-discuss calibration
-discuss functions were made from scratch, not from libraries

Trajectory Planning and Following

The first method we use for robot path creation is Trajectory Planning via Cartesian Polynomials as outlined by Dr. Gian Luca Mariottini [1]. We first create a path based on an initial position, initial orientation, final position, and final orientation, all of which exist in the image plane. The trajectory equations are then parametrized depending on the desired number of points along the path. The result is a set of points, each of which include a horizontal, vertical, and angle component, that all together create the total desired path in the image plane for which the robot can follow to travel from the initial point to the end point.

In order for the iRobot Create to reach and maintain the desired path, a control algorithm needs to be implemented to provide input to the robot. Again, Dr. Mariottini has designed a control algorithm that outputs translational velocity and angular velocity of the robot [3]. This negative feedback system has an error which is defined as the input, the current robot beam position, minus the output, the desired trajectory beam position. This controller acts as a simple proportional (P-type) controller and intends to make the error approach zero exponentially.

The intention of following a path is for the robot to arrive at the desired trajectory point and the robot's current position to match the desired position for subsequent trajectory points. As discussed in the previous section, the robots current orientation and position is calculated from the video feed. Due to the iRobot Create being modeled as a unicycle robot, the input to the control system must be a position and orientation so the control output signal is velocity, or in other words a speed with a heading. To represent a heading for the robot, a virtual vector, or beam, of a set length is drawn from the robot's current position along the robot's current orientation. A second virtual vector of the same length as the first is drawn from the current desired trajectory point along the desired trajectory orientation. Each of these beams can seen in Figure 4.

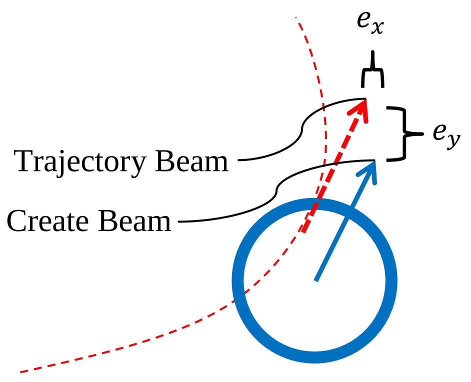
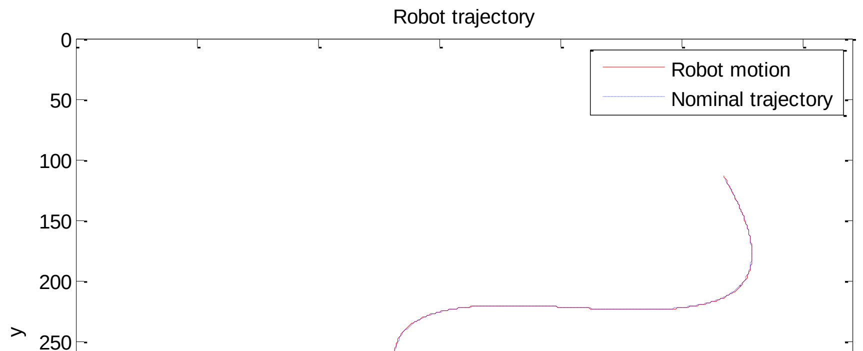


Figure 4 Robot and Trajectory Beam Diagram

The object of the controller is to make both beam positions match, with each beam point representing a heading at a set length away from the current position. Since the robot beam and trajectory beam are the same length, when the robot beam point matches the trajectory beam point, the robot's current position and orientation equal the desired trajectory position and orientation. The controller attempts to keep the robot beam position and trajectory beam position equal to each as the robot travels along the path by varying the output translational and angular velocities.

There are multiple constants required for trajectory planning, trajectory tracking, virtual beam creation, and controller gains. All of these numbers were initially found by using Matlab to simulate the trajectory and trajectory tracking. The simulation results are shown in Figure 5.



Point-to-Point Motion Control

As a simple position control method, we also implement a point-to-point motion control technique using a proportional controller. The control error is defined to be a vector of two elements. The first error components is the Euclidean distance to the target point whereas the second one is the orientation of the target point relative to robot position and orientation as illustrated in Figure 6.

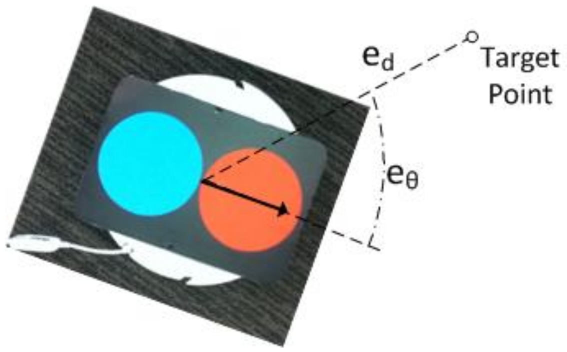


Figure 6 Definition of control errors for point-to-point motion control

The point-to-point control algorithm stops the robot as soon as the robot distance to the target point is less than a threshold value. This ensures that the target point is not bypassed and the orientation error does not change abruptly. The proportional control law is given as:

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} K_v & 0 \\ 0 & K_w \end{bmatrix} \begin{bmatrix} e_d \\ e_\theta \end{bmatrix} \tag{2}$$

where K_v and K_w are the control gains and e_d and e_θ are the control errors. The control code automatically determines the direction to turn to given the pose of the robot and the position of the target point.

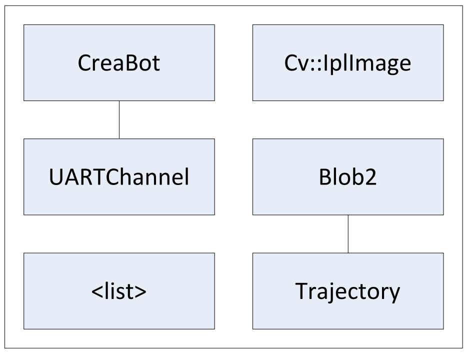
C++ Software Implementation

We have used Microsoft Visual Studio 2010, C++, and OpenCV 2.3.1 to code the robot control program. There are several class structures that we developed as shown in Figure 7. Brief descriptions about these classes are as follows:

- UARTChannel Class: This class is used to establish serial communication with the robot. It includes transmit and receive buffers and methods to open and close a comport, send and receive bytes.
- CreaBot Class: This is a basic robot control class that implements the communication with the actual robot through a UARTChannel object. This class also includes a proportional control method that implements the control law in eq. (2).
- Blob2 Class: The detection of the red and blue blobs is performed by this class. It includes trajectory structure as a member so that the desired trajectory is defined in it.
- Trajectory Structure: The trajectory parameters are stored and updated using this structure.

Some other classes we use are the Cv::IplImage class for image processing and the Microsoft list structure to store experimental data in a dynamic list. The list structure is used to store experimental data.

The control of the robot is performed using a finite state machine implemented in the main loop of the program. The state diagram shown in Figure 7 includes 8 states except the mouse event handlers for user interaction. The first four states are preparatory states in which the user is asked to provide a comport number for serial connection, to make selection of either the trajectory tracking control or the point-to-point control options, and then to make the selection of the red and blue regions in the image for calculation of hue and saturation thresholds. Afterwards, the program moves into the ready state where the endpoint and end-orientation of the trajectory is obtained from the user. Then, the calculated trajectory is plotted on the screen for a second and the robot starts moving. In the trajectory tracking control case, the robot follows the trajectory. In the point-to-point motion control case, the robot does not follow the trajectory but moves to the endpoint regardless of the intermediary points. The robot stops when it reaches the target point or the user hits spacebar.



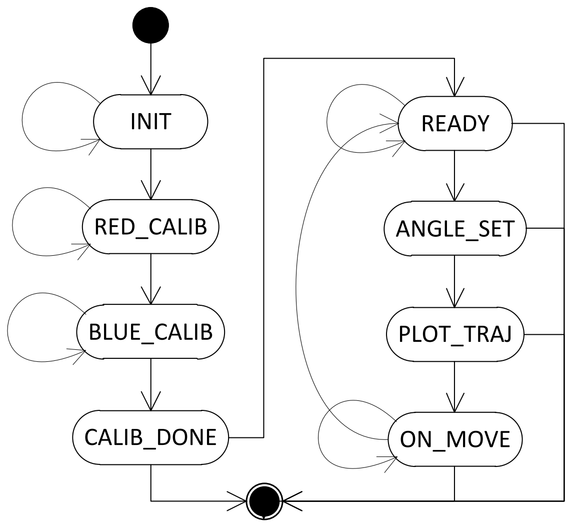
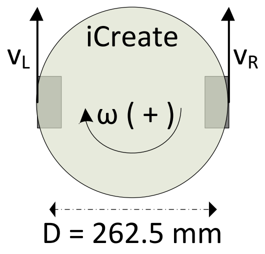


Figure 7 Class and state diagrams of the implemented C++ code

Direct Drive

The iCreate robot has two drive modes one of which is the direct drive. It is convenient to be used for steering the robot along a path as the control signals v and ω can easily be converted to the direct drive velocity inputs. Assume that vR and vL are the right and left wheel velocities (mm/sec); and v and ω are the translational (mm/s) and rotational (rad/sec) velocities of the robot, respectively. D is the diameter (mm) between two wheels. The control signals, v and ω, are related to the direct drive velocities as follows:

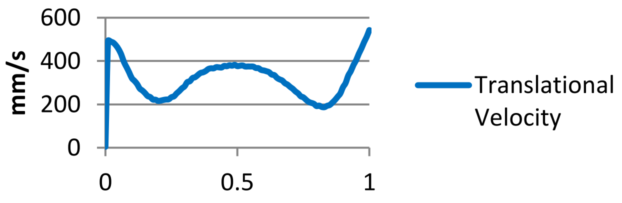


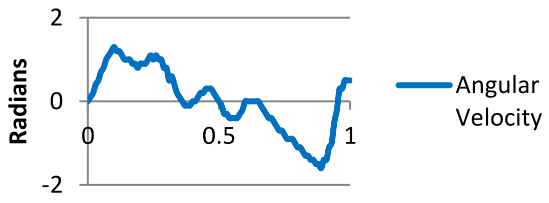
$$\begin{bmatrix} v_R \\ v_L \end{bmatrix} = \begin{bmatrix} 1 & -D/2 \\ 1 & D/2 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Figure 8 Control velocities to direct drive conversion

Experimental Results

In order for the simulation and experimental results to be comparable, the test scenario kept the initial and final robot orientation the same for all three cases. In its current iteration, the trajectory planning and following control system yields functional and stable results with room for future improvements. There are numerous system parameters and controller gains that are currently constants, and could be improved by implementing algorithms to further minimize error. The system performance can be seen in Figure 9.





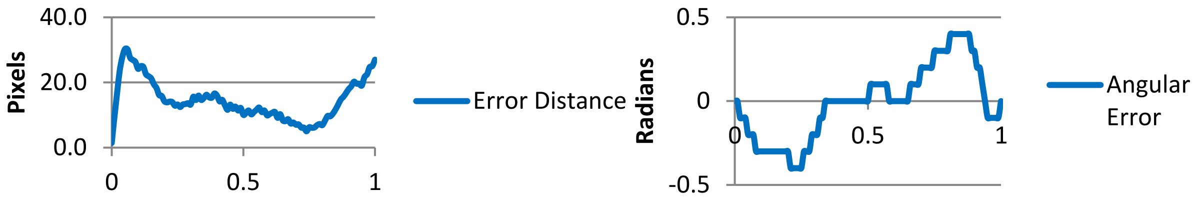


Figure 9 Trajectory Tracking System Performance

Here it is important to note that the "Distance Error" calculated is defined as the distance from the robot beam point to the current desired trajectory beam point, and the "Angular Error" is defined as the difference between the robot's current orientation and the desired trajectory orientation. Trajectory tracking provided very favorable results from the test scenario with the Create following the desired path with minimal oscillations. In Figure 10, the planned trajectory is shown as the green line while the path the robot traveled is in red.

For the Point-to-Point motion control, the errors are fundamentally different due to the definition. In Point-to-Point, the distance error is defined as the Euclidean distance between the robot's current position and the end point. Since this method does not define a set trajectory for the robot to follow, there are no trajectory points for which the same distance error can be calculated as in the trajectory tracking method. Point-to-Point controller performance can be seen in Figure 11.

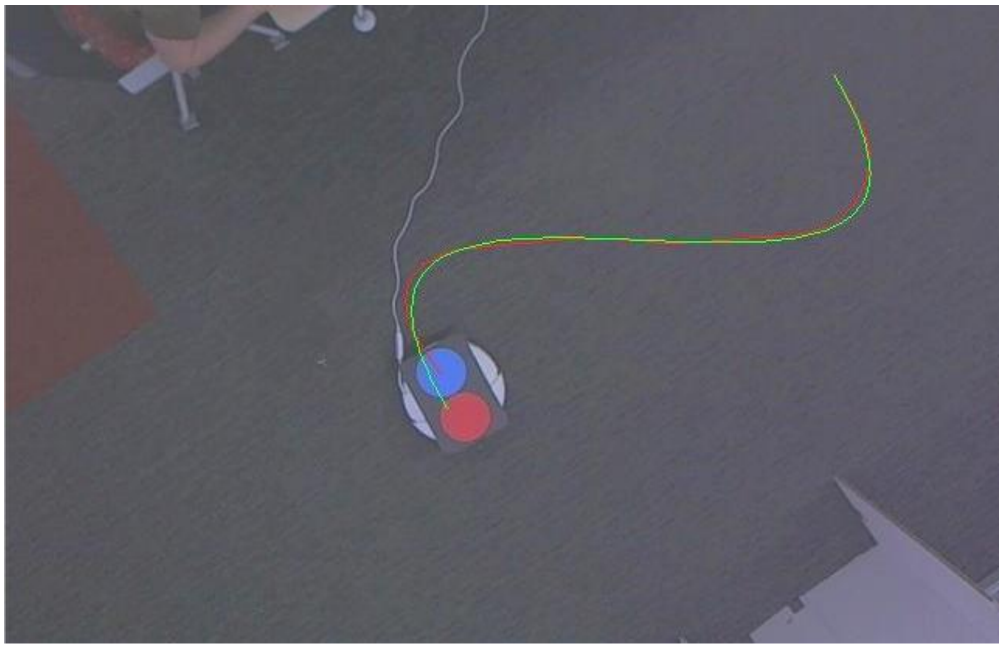


Figure 10 Trajectory Tracking Test Results

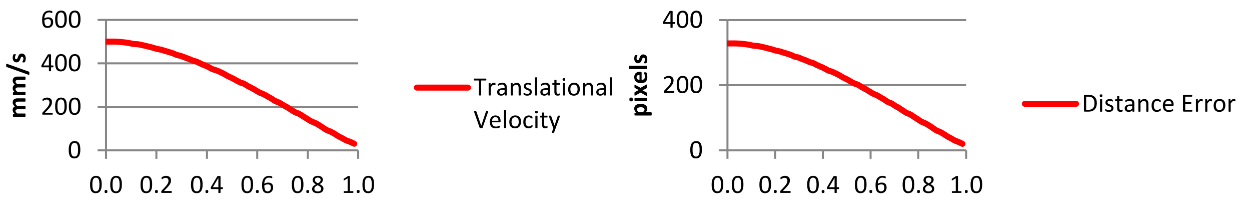


Figure 11 Point-to-Point System Performance

There are multiple differences between these two implementations of motion control. For trajectory tracking, the robot's desired trajectory is known before movement begins while for Point-to-Point, the robot's trajectory is unknown and simply a result of the controller minimizing the orientation and distance from the robot's current location to the desired point. Trajectory tracking is more complex than Point-to-Point due to the possibility of the robot outpacing or falling behind the current trajectory point. One of the major disadvantages of Point-to-Point is the final orientation is not set by the user, but instead a function of the robot's traveled path to the final point. While each of these systems provide the same overall function, they each have advantages suitable for various applications.

Conclusions and Future Work

The current detection algorithm of red and blue blobs is sensitive to changes in the camera properties. That results in the requirement of selecting a larger interval between the hue and saturation thresholds of the two colors. In turn, that causes the detection of false colors such as the red in the carpet. This could be eliminated by performing the detection process on a region of interest that tracks the position of the robot in the image. Also, more advanced filtering algorithms could be employed to choose the correct color segment in the image.