CSE276A – HW2

Luke Liem
A5231779

Abstract

In this assignment, the JetBot needs to follow a path traversing 2 way-points before returning back to the origin. Using a combination of IBVS as well as simple rotate-and-detect, I successfully programmed the robot to travel from (0,0,0) to (1,0,0), then from (1,0,0) to (1,2,π). However, due to space constraint and lack of suitable large reference objects, the return of the robot from (1,2,π) back to (0,0,0) was not attempted. (https://youtu.be/UOumUupGDgg)

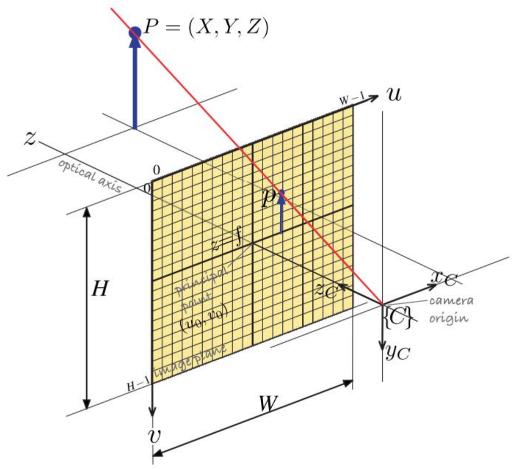
Due to the low resolution of the video output (300x300 pixels), the Single-Shot Detector (SSD) object recognition model has very limited object detection range. For example, if we want the robot to move 2m forward, we need to place a 1st large object (e.g. a chair) 2m from the starting point to guide the robot in its 1st one meter forward; then we need to place a 2nd large object (e.g. large plant) 3m from the starting point to guide the robot in its next one meter forward.

Nevertheless, the experiment demonstrates that IBVS can guide the robot fairly accurately to a target Cartesian coordinate at precision of a few centimeters, ***provided*** there are enough reference objects placed strategically along its path recognizable by SSD object recognition. However, IBVS is unsuitable for guiding the robot to a correct orientation. This is because any orientation adjustment of >45° would push the reference object's feature points outside the video frame, causing the IBVS control to fail.

In its place, I have implemented a simple rotate-and-detect algorithm which makes the robot left-rotate in place in tiny small steps until a strategically place reference object appears near video image's focal center. This algorithm is capable of guiding the robot to its correct orientation at precision of 5-10°.

Background

Consider a camera observing a world point *P* with camera relative coordinates *P = (X, Y, Z).* Let $\bar{u} = u - u_0$ and $\bar{v} = v - v_0$ be the pixel coordinates of *P* relative to the image plane's principal point along the X and Y directions.



The relationship between the velocities of the pixel coordinates $(\dot{\bar{u}}, \dot{\bar{v}})$ and the camera's velocities in the camera's reference frame is defined by the following equation:

$$\begin{pmatrix} \dot{\bar{u}} \\ \dot{\bar{v}} \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{f}{\rho_u Z} & 0 & \frac{\bar{u}}{Z} & \frac{\rho_u \bar{u}\bar{v}}{f} & -\frac{f^2+\rho_u^2\bar{u}^2}{\rho_u f} & \bar{v} \\ 0 & -\frac{f}{\rho_v Z} & \frac{\bar{v}}{Z} & \frac{f^2+\rho_v^2\bar{v}^2}{\rho_v f} & -\frac{\rho_v \bar{u}\bar{v}}{f} & -\bar{u} \end{pmatrix}}_{J_p} \begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

In this assignment, since the SSD Objection Detection software already generates a bounding box for each detected object, the 4 corners of the bounding box conveniently provide the 4 feature points necessary for IBVS control:

$$\begin{bmatrix} \dot{\bar{u}}_1 \\ \dot{\bar{v}}_1 \\ \dot{\bar{u}}_2 \\ \dot{\bar{v}}_2 \\ \dot{\bar{u}}_3 \\ \dot{\bar{v}}_3 \\ \dot{\bar{u}}_4 \\ \dot{\bar{v}}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{J_p^1} \\ \boldsymbol{J_p^2} \\ \boldsymbol{J_p^3} \\ \boldsymbol{J_p^3} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \boldsymbol{L} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \qquad (1)$$

If we set the robot's relative reference frame to be the same as that of the camera, the relationship between the camera velocities and the differential-drive robot's velocities is defined by the following equation:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} \qquad (2)$$

$$\Rightarrow v_z = v_d;\ \omega_y = \omega_d; v_x = v_y = \omega_x = \omega_z = 0$$

Furthermore, the relationship between the robot's velocities in its relative reference frame (v_d, ω_d) and the robot's wheel angular velocities (ω_r, ω_l) is defined by the following equation:

$$\begin{bmatrix} v_z \\ \omega_y \end{bmatrix} = \begin{bmatrix} {r_w}/{2} & {r_w}/{2} \\ {r_w}/{L} & -{r_w}/{L} \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} = \boldsymbol{T} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} \tag{3}$$

where L is the axle length and r_w is the wheel radius.

Within this context, IBVS is simply a linear controller that drive the feature points toward their desired values on the image plane based on the following equation:

$$\begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} = gain * \boldsymbol{T}^{+}\boldsymbol{H}^{+}\boldsymbol{L}^{+}(\boldsymbol{p}^{*} - \boldsymbol{p}) \tag{4}$$

where ***p**** is the vector of the pixel coordinates of the reference object's 4 feature points when the robot reaches its target pose, and ***p*** is the vector of the pixel coordinates of the reference object's 4 feature points when the robot is at its current pose.

Implementation Tricks

Due to the lack of encoder feedback and the unpredictability of the PWM motor control on the Jetbot, I have to implement these 3 tricks in order to guide the robot reliably with IVBS and rotate-and-detect:

1. Stepwise motion
2. Translational and angular velocity clamping
3. Strategic placement of reference objects

Stepwise Motion

By varying the speed settings of the robot's motors from 0.1 to 0.9, and measuring the distance traveled by the robot in 0.1 second, I obtained a table that maps motor setting to wheel angular velocity.

Motor Setting	On time (s)	Distance (m)	m/s	ω
0.25	0.1	0.000	0.000	0.000
0.3	0.1	0.013	0.125	3.846
0.4	0.1	0.030	0.300	9.231
0.5	0.1	0.049	0.488	15.000
0.6	0.1	0.084	0.838	25.769
0.7	0.1	0.095	0.950	29.231
0.8	0.1	0.115	1.150	35.385

My experience with JetBot's unreliable PWM motor control in Homework 1 convinced me to forego continuous motion control and implement stepwise motion control instead. Stepwise motion minimizes the chance that unreliable control of wheel motor speeds would lead to sudden and unexpected large orientation change which can push the reference object outside of the camera's video frame, thereby incapacitating IBVS control:

```
robot.set_motors(w_l, w_r)  # left, right
time.sleep(Rtime)  # 0.1 second
robot.stop()
```

Translational and angular velocity clamping

Keeping the reference object's feature points (bounding box) within the camera's video frame is critical for IBVS control. In addition to step-wise motor, I have implemented a clamp () function which places limits on the robot's translational and angular velocities:

- Clamp translational speed between a minimum of 0.14 m/s (equivalent to wheel motor setting of 0.3) and a maximum of 0.20-0.30 m/s (equivalent to wheel motor setting of approx. 0.4)
- Clamp angular speed to a maximum of 0.5 radian/s (equivalent to 2.9° per 0.1 second step time)

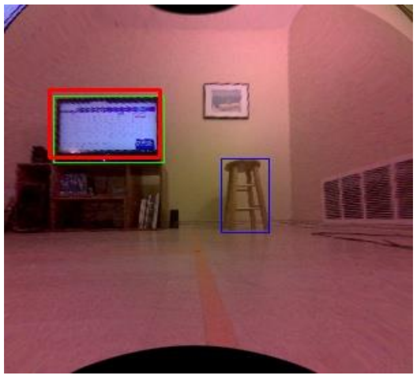
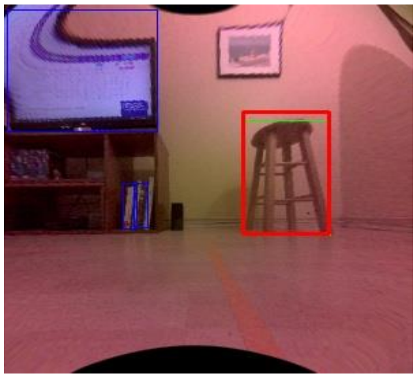
Strategic placement of reference objects

Suitable reference objects need to be placed strategically along the robot's intended path so that the IBVS and rotate-and-detect algorithms can use them to guide the robot to its correct poses.

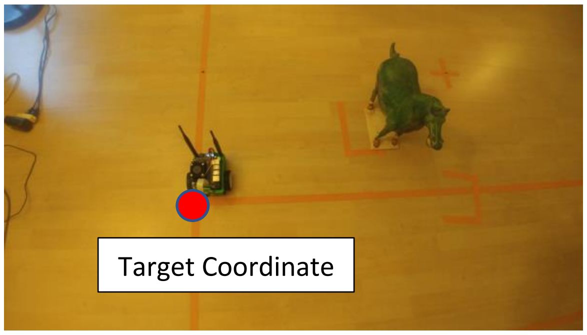
Reference Object	*Need for*	*Purpose*
Large TV	IBVS	Guide robot from (0,0,0) to approx. (0.5,0,0)
Stool	IBVS	Guide robot from (0.5,0,0) to (1,0,0)
Horse Statue	Rotate-and-Detect	Orient the robot to approx. (1,0,π/2)
Chair	IBVS	Guide robot from (1,0,π/2) to approx. (1,1, π/2)
Large Plant	IBVS	Guide robot from approx. (1,1, π/2) to (1,2, π/2)
Computer Mouse	Rotate-and-Detect	Orient the robot to approx. (1,2,π)

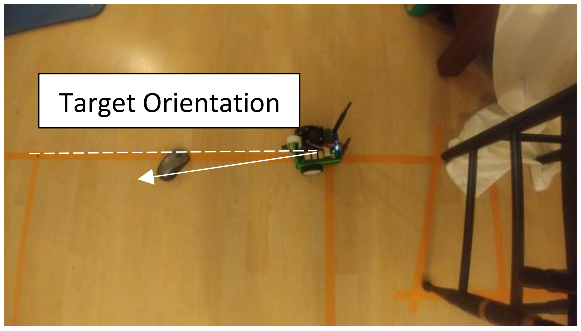
Results

In the majority of the test runs, the robot successfully executed the path from (0,0,0) to (1,2,π). Implementation of stepwise motion limited control loop update at 3.0-4.0 FPS. IBVS control appeared to be working well, successfully guiding the bounding box of the reference object (green) towards the target bounding box (red) in most cases.

Precision overall is acceptable: within a few centimeters of the target Cartesian coordinate for IBVS and within 5-10° of the target orientation for detect-and-rotate:





A video of a test run is available on YouTube: https://youtu.be/UOumUupGDgg

Conclusion

IBVS can provide fairly accurate guidance of a robot to a target pose even without encoder feedback. However, the method is severely limited by the robot's object detection range and it cannot guide the robot to make large orientation change. This necessitates the need for a host of implementation tricks like rotate-and-detect, motion stepping, velocity clamping and the placement of many small and large reference objects along the robot intended path.

Reliability and effectiveness of IBVS should increase with a more powerful GPU and a higher-end camera, which would improve both range and accuracy of the SSD object detection algorithm.