Trajectory Following on the PRG Husky

Team `sudo rm -rf *`

Abhishek Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: shastry@umd.edu

Animesh Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: animeshs@umd.edu

Nicholas Rehm
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: nrehm@umd.edu

***Abstract*—In this report a controller for trajectory following on the PRG Husky is implemented and tested for ENAE788M: Hands on Autonomous Aerial Robotics. A closed loop PD controller was designed utilizing feedback from onboard odometry readings in order to more accurately follow either continuous time-parameterized trajectories or a set of waypoints. The results of this control scheme lead to far better trajectory following compared to an open loop scheme. The method can easily be applied to new trajectories in the future.**

I. Introduction

In order for aerial robots to perform useful tasks, they must be able to follow prescribed trajectories accurately and (ideally) quickly. An outer-loop control scheme must be implemented to command the vehicle's inner-loop controller to perform the required maneuvers to achieve a desired trajectory. This can be done either with an open loop scheme, where position or orientation commands are generated independent of the current vehicle state, or with a closed loop scheme, where commands are generated with some form of state feedback. In some cases, an open loop strategy may be acceptable to achieve a change in vehicle position or orientation. However in other cases where precise motion and position is required, a closed loop strategy is needed.
Link to the result videos: Click Here

II. Open Loop

An open loop control scheme was considered to follow the required helical, diamond, and step trajectories. The bebop's internal controller varies the vehicle's velocity in the x, y, and z directions for a specific time in order to change the vehicle's position. This can only be done in increments of one 'unit,' which corresponds to 1 meter on the stock bebop platform. Calibration on the PRG Husky found that commanding 1 unit in any every direction leads to a displacement of about .25 meters in the corresponding direction. Therefore using this method to command the vehicle along the desired trajectories would require it to traverse and stop every .25 meters which is not ideal especially for continuous trajectories. A solution to this is to time parametrize the trajectories and use an approach in which velocity is continuously commanded along the desired trajectory for prescribed times. This method would require a copious amount of tuning and would be susceptible to severe error accumulation. Additionally, open loop control scheme are not at all robust against external disturbances. A slight wind gust will severely degrade the open loop control's performance.

III. Closed Loop

A closed loop control scheme is much better suited for accurate trajectory following and to prevent significant error accumulation over time. It is also much safer to test and perform the desired maneuvers because the vehicle location is generally known throughout a maneuver. A closed loop control scheme used in completing the required trajectories is described below.

A. Feedback

The feedback data is available in the form of odometery messages on the topic `/bebop_driver/odom` at a frequency of around 5Hz. The message contains the position, velocity, and orientation information obtained from various internal sensors in the bebeop frame. Position data must be scaled by a 'terrain constant' which is a linear scale factor calibrated to the floor of the test area. The position and the velocity data are only used as feedback to the external controller. In the future, an EKF will be implemented to fuse the velocity and acceleration (obtained from the orientation data) to get a filtered position and velocity feedback at higher frequencies than 5Hz.

B. Controller

A Proportional-Derivative (PD) controller has been implemented as a Python function inside a ROS node. The control commands are essentially reference velocity signals which the internal controller of the bebop has to track. The output of the controller is published on the topic `/bebop_driver/cmd_vel` at 10Hz. The following equation describes the implemented PD control structure.

$$\boldsymbol{u} = K_P\left(\boldsymbol{x_d} - \boldsymbol{x}\right) - K_D\boldsymbol{v} \tag{1}$$

where x_d is the reference position commands, x is the current position and v is the current velocity and K_P and K_D are the

proportional and derivative gain matrices respectively.
As the odometery feedback is slightly distorted and the amount of distortion varies with the type of patterned surfaces that the quadrotor flies over, we define a surface scaling factor k_{surf} for the feedback data. Now, the final corrected control expression with feedback correction is described below.

$$\boldsymbol{u} = K_P\left(\boldsymbol{x_d} - k_{surf}\boldsymbol{x}\right) - K_D k_{surf}\boldsymbol{v} \tag{2}$$

Controller gains K_P and K_D were tuned through a series of successive flight tests. A less aggressive set of gains were initially chosen and progressively increased until the vehicle no longer tracked the commanded trajectories in a desirable manner. Oscillations in the tracking response with more aggressive gains were solely the result of the odemetry feedback being limited to 5Hz. Therefore the controller has been optimally tuned for the fastest and most accurate performance for a particular surface pattern.

C. Reference commands

The position reference signals are either supplied as a continuous time-parameterized trajectory or a set of waypoints. When waypoints are supplied, an extra piece of code is executed which essentially allows the current waypoint to switch to the next waypoint only when the norm of the current position error and velocity error of the vehicle comes within a predefined parameter, called acceptance radius for the position error and acceptance velocity for the velocity error. This ensures that the vehicle navigates to all of the waypoints sequentially and accurately. Without the acceptance conditions for each waypoint, the trajectory tracking suffered because vehicle drift after reaching one waypoint would propogate into the trajectory toward the next waypoint. For example, rather than tracing sharp steps in the step trajectory, a lack of waypoint acceptance conditions caused each sequential step to overshoot due to vehicle drift from reaching the previous waypoint.

D. Trajectory Interpolation

Due to the coupled nature of the quadrorotor's dynamics, the tracking performance in the x-y plane is lower than that in the z-axis. Hence, if we wish the quadrotor to get to waypoints in a straight path then we need to implement trajectory interpolation between successive waypoints. This is generally done by defining a polynomial in time of some order based on the minimization of the n^{th} differential of the flat outputs (for differentially flat systems), with boundary conditions applied on all the $n-1$ flat outputs. For $n=3$, a minimum acceleration trajectory can be generated by solving the coefficients of a 3^{rd} order polynomial. Similarly, for minimum jerk $n=4$ and minimum snap $n=5$ the trajectory is of 5^{th} order and 7^{th} order in time respectively. For example, in case of minimum acceleration the trajectory for each of the flat output $x(t)$ is represented as follows.

$$\begin{aligned}
x(t) = \\
&t^2\left(\frac{3x_0(t_0+t_f)}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3} - \frac{3x_f(t_0+t_f)}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3}\right) \\
&-t\left(\frac{6t_0t_fx_0}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3} - \frac{6t_0t_fx_f}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3}\right) \\
&-t^3\left(\frac{2x_0}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3} - \frac{2x_f}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3}\right) \\
&+x_0\frac{-t_f^3+3t_0t_f^2}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3} - x_f\frac{-t_0^3+3t_ft_0^2}{t_0^3-3t_0^2t_f+3t_0t_f^2-t_f^3}
\end{aligned}$$

where x_0 and x_f are the boundary conditions for the position at $t=t_0$ and $t=t_f$ respectively. The velocity boundary condition has been set to zero. During the testing of the minimum acceleration trajectory interpolation the quadrotor crashed. In the future more testing will be done if this method is needed.

IV. Results

A. Steps

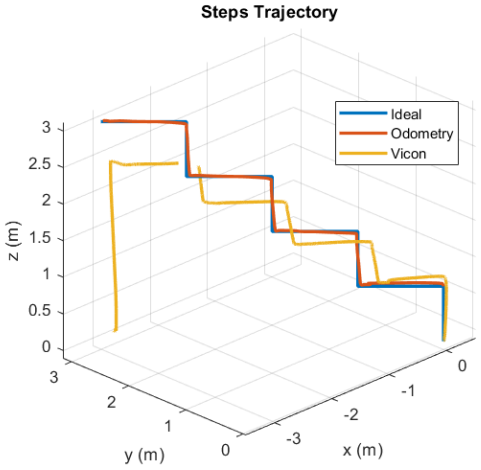

Fig. 1. Isometric view for the Steps Trajectory

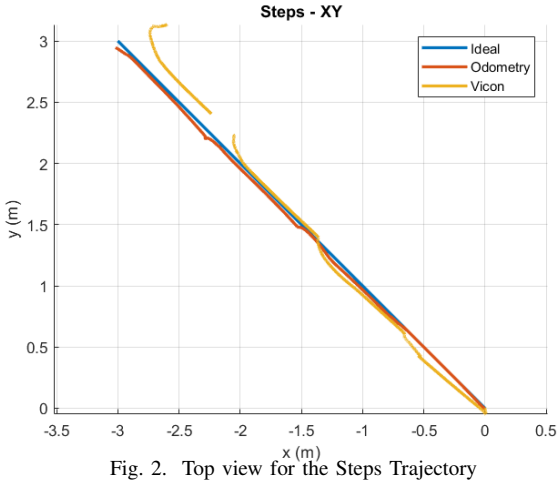

Fig. 2. Top view for the Steps Trajectory

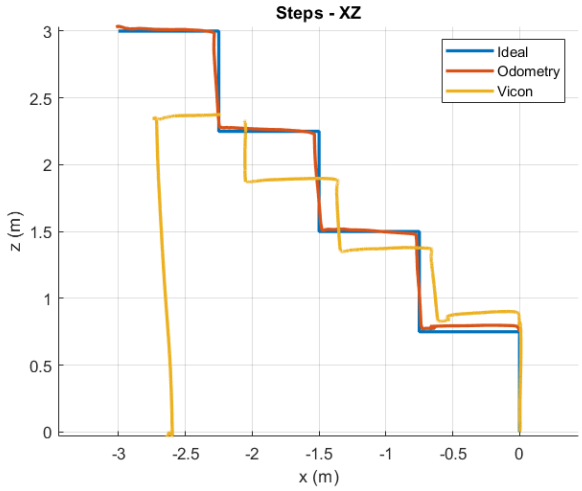

Fig. 3. Front view for the Steps Trajectory

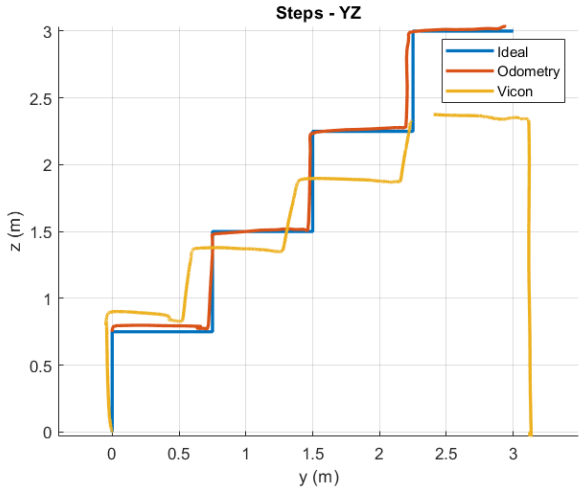

Fig. 4. Side view for the Steps Trajectory

B. Diamond

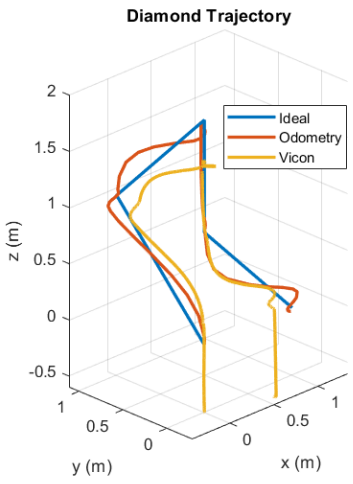

Fig. 5. Isometric view for the Diamond Trajectory

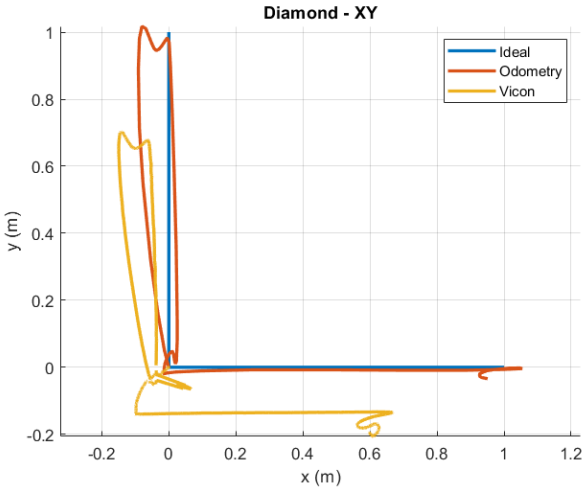

Fig. 6. Top view for the Diamond Trajectory

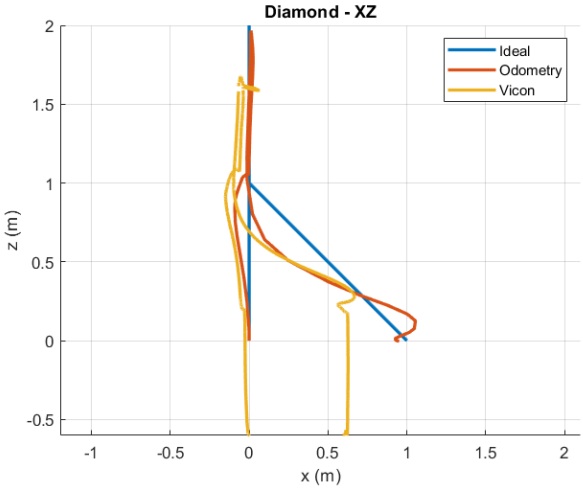

Fig. 7. Front view for the Diamond Trajectory

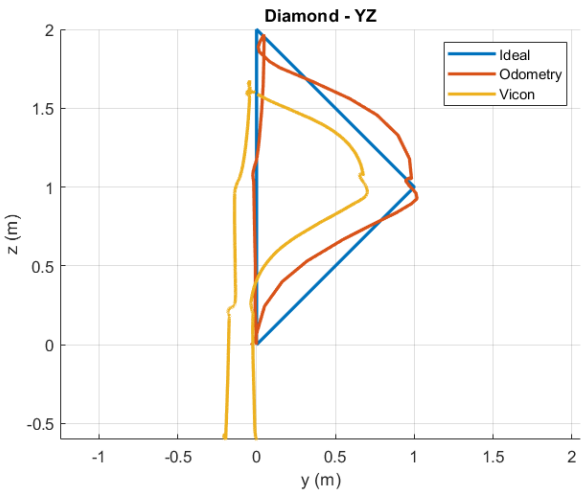

Fig. 8. Side view for the Diamond Trajectory

C. Helix

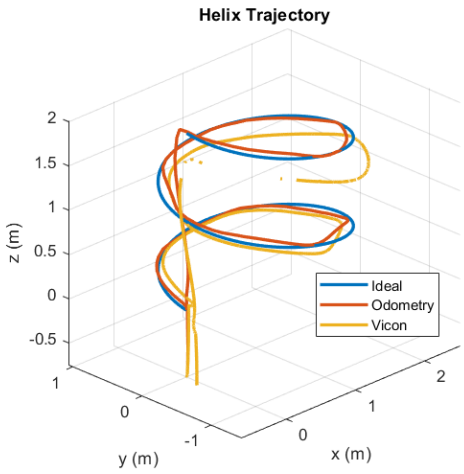

Fig. 9. Isometric view for the Helix Trajectory

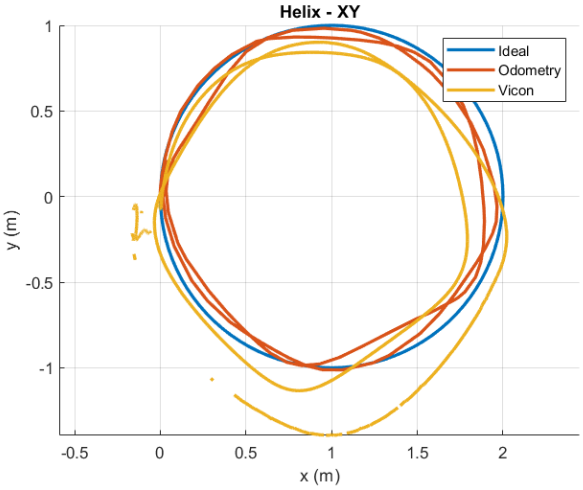

Fig. 10. Top view for the Helix Trajectory

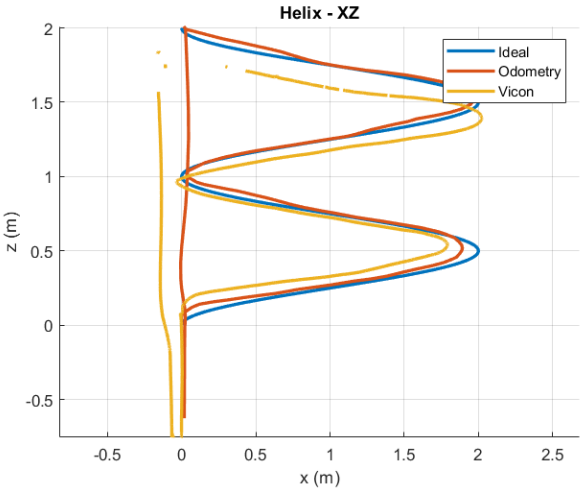

Fig. 11. Front view for the Helix Trajectory

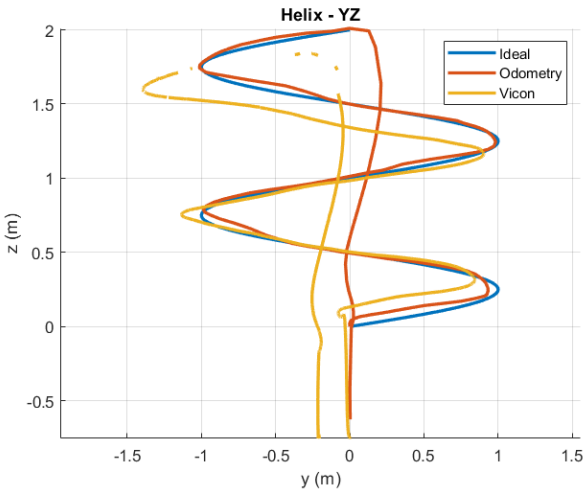

Fig. 12. Side view for the Helix Trajectory

V. Conclusion

The trajectory which the vehicle follows is scaled down but similar in shape and form to that of the reference trajectory. This is due to the fact that the surface over which the Vicon plots were generated and the surface over which the parameter k_{surf} was tuned are different. The Bebop's odometry is only reliable when it is properly scaled for a particular surface. However, the odemetry trajectory matches the ideal trajectory very well meaning the controller was tuned well. With vicon truth data now available, the surface parameter can be precisely tuned so that the actual trajectory matches the ideal trajectory. In the future, the velocity part of the Bebop's odometry will be scaled and fused with the Stereo odometry from the Tara Camera to get better pose estimates.