Trajectory Controller for a Bebop Quadcopter

Ilya Semenov
University of Maryland
isemenov@umd.edu

Tim Kurtiak
University of Maryland
tkurtiak@terpmail.umd.edu

***Abstract*—This report presents the implementation and results of a closed loop trajectory controller for a Parrot Bebop quadcopter. The controller was built as an outer loop to expand upon the Bebop's attitude and position controller. Three trajectories, Helix, Diamond, and Stairstep were executed.**

I. Problem Statement

This project aims to implement an outer loop trajectory controller for the Parot Bebod quadcopter using Robot Operating System. In addition, the telemetry is visualized and compared to the desired trajectory using VICON data plotted in RVIZ.

II. Trajectory Parameterization

In order to implement a trajectory controller, we must first generate a feasible trajectory. Three trajectories were considered in this project: a Diamond, a Helix, and a Staircase described in Figure 1 2 , and 3

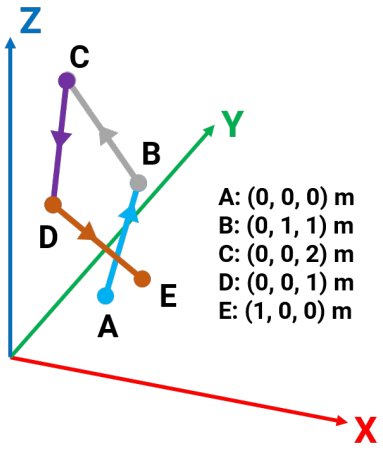

Figure 1. Diamond Trajectory

The diamond trajectory is parameterized using target waypoints described by the points in figure 1. The staircase trajectory requires an angle of 45 degrees between the path and the vector $< 3, 3, 3 >$, connecting several colinear points that are along that vector as shown in 3. This requirement along with 5 total colinear points forces the waypoints for this trajectory. Lastly, the helix trajectory is defined continuously in figure 2. In this implementation 1 revolution is discretized into 10 waypoints that satisfy the equations:

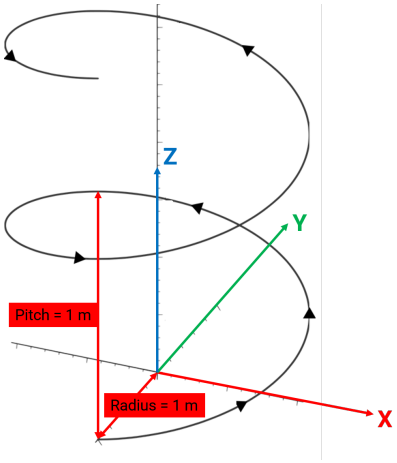

Figure 2. Helix Trajectory

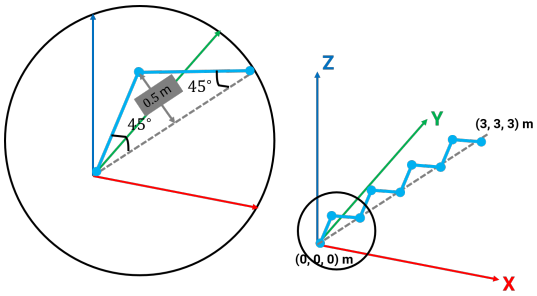

Figure 3. Staircase Trajectory

$$\begin{cases} x = sin(t/2\pi) \\ y = cos(t/2\pi) \\ z = t \\ t \in [0, 1] \end{cases} \tag{1}$$

This describes one rotation of a helix of radius one and height one.

With each trajectory represented as a series of waypoints, these waypoints are then sent to the outer loop controller as setpoints in order. It should be noted that all of these waypoints are with reference to the waypoint frame, which is an inertial frame.

III. Design of Outer Loop Controller

The outer loop controller must steer the quadcopter to a desired setpoint while taking feedback from the quadcopter's position sensor. This is achieved through a closed loop tracker controller based on position and velocity error.

First it must be noted that the Bebop drone assigns an inertial frame upon being turned on, and all odometry readings are collected relative to that frame. However, the commands sent to the drone are in the body frame and waypoints are described in the waypoint frame. An open loop controller can suffice with a series of commands set purely in the body frame, but to use odometry information as feedback a rotation must be preformed.

The orientation of the body frame B is encoded in a unit quaternion. Let q_{BI} be the unit quaternion that describes the rotation to the body frame from the inertial frame I. Then a vector v_I in the inertial frame is made into a non-unit quaternion $q_{vI} = [0, v_I]$ and transformed into non-unit quaternion $q_{vB} = [0, \bar{v_B}]$ via:

$$q_{vB} = q_{BI} \otimes q_v \otimes {q_{BI}}^{-1} \tag{2}$$

Where $\otimes$ denotes a quaternion multiplication. Vector $\bar{v_B}$ is a representation of v_I in the inertial frame. However, the translation of the body frame needs to be taken into account before the final vector v_B is constructed.

It should be noted that ${q_{BI}}^{-1} = q_{IB}$ and the rotation of a vector from the body frame into the inertial is the same process as above.

Way points are described in the waypoint frame W which is a fixed frame equal to the body frame at a user specified time after takeoff with orientation $q_W I$ and displacement rW_I represented in the inertial frame. Let a waypoint in the waypoint frame w_W be given. The controller must first rotate the waypoint into the inertial frame via vector rotation by $q_I W$, resulting in $\bar{w_I}$. Then the location of the waypoint in the inertial frame is $w_I = \bar{w_I} + rW_I$.

The error vector in the inertial frame is $e_I = w_I - rB_I$ where rB_I is the current displacement of the body frame origin represented in the inertial frame.

The error vector is rotated by $q_B I$ to result in e_B and it is a function of time t. Translation is taken into account pre-rotation.

Given that the waypoint was assigned at time $t = t_0$ then the control command u at time t is given:

$$u = K_p . * e_B + K_i . * \int_{t_0}^{t} e_B dt - K_d . * \frac{e_B}{dt} \tag{3}$$

Where $.*$ represents element wise multiplication and K_p, K_i, K_d are gain vectors.

IV. Implementation

The implementation is structurally simple, but many tests were carried out leading up to the final point. A python script functions as a node with several publishers and subscribers. A subscriber to topic bebop/odom receives odometry data and saves it to global variables. A publisher to bebop/takeoff sends an Empty message for the vehicle to take off. After a pause, the odometry data is saved as a separate global variable, this is now the basis of the waypoint frame.

A function moveto() was created that is passed waypoint coordinates (in the waypoint frame) calculates the inertial frame representation of the way point, and proceeds to execute the loop culminated by equation 3. The output u is sent as a Twist message to topic $bebop/cmd\_vel$. The components of u are placed in the linear portion of a Twist message only, there is no yaw in this system at this time.

The waypoints that are passed to moveto() are the only thing that changes between scripts for different trajectories. The gains are also subject to change.

One major challenge of this project was the lack of physical intuition regarding publishing Twist messages to the $bebop/cmd\_vel$ topic. The combination of values and time delay post command change the behavior of the vehicle upon execution considerably. Additionally, the responses in the x,y,z directions are not consistent amongst themselves. These challenges were found upon initial attempts for an open loop controller, and where the primary driving factor for closed loop.

Another issue that was identified late and not rectified fully was the uncertainty in odometry data given by the vehicle. Especially at low altitude this data would be very erroneous, and would build up error over longer flights.

Special thanks to Abhishek for advice throughout development and for lending a hand during the demo, Animesh for help during the demo, and Derek Thompson for providing a template for RVIZ.

V. Results

Result videos are posted here:

helix https://youtu.be/0wlfIFKXbyQ diamond https://youtu.be/e3$_q xK9pr8Astairhttps$: $//youtu.be/D381VyEMNko$

Despite the multiple crashes that occurred during the demo, there is reason to be optimistic about the performance of the Bebop. Notably, it tracked it's own odometry data well.

Inspecting the staircase trajectory (trajectory 3) shown in figure 6 one can see that the odometry data valiantly follows the waypoints, meeting at each vertex. Some minor issues exist when only considering the odometery, namely there is slight overshoot in the x-y plane. This causes the trajectory as seen by the odometry to bend out slightly from the straight vertical segments.

The caveat here is that the vicon data does not support the odometry readings. Examining the X-Z and Y-Z views it is clear that the vehicle tends to drift in the X-Y plane without accounting for it in odometry readings, This causes the absolute distance between the "corners" of the staircase as measured by odometry and vicon data to increase with time. An accumulation of error over time of this magnitude is unacceptable for control implementation and is likely an issue with the downward facing optical flow camera.

The odometry Z-axis readings do not exhibit this same level of drift, which can be assumed to mean that the down facing sonar data is more dependable than the optic flow camera data regarding position.

Inspecting the Helix trajectory (trajectory 1) shown in figure 4 a similar conclusion is reached. While the vehicle odometry shows good tracking of waypoints, the odometry is not accurate. The Z-axis odometry readings once again show adequate correlation with vicon, however the X-Y readings are completely off. Looking at the X-Y view it is clear that the vicon data drifts further away from odometry with time. Again, the vehicle does not record it's movement in the X-Y plane when it does indeed move. This issue was the main cause of the first crash of this trajectory.

Finally, inspecting the Diamond trajectory the same conclusions can be reached. However this trajectory illustrates one of the flaws with the algorithm that isn't tied to faulty odometry readings. It is clear here that the vehicle does not follow straight line paths between nodes due to conserved velocity upon reaching a node. If percise movement is needed this velocity could lead to crashes between waypoints even if odometry readings are perfect. A simple fix is implementable. Consider the waypoint reached if the vehicle is within some error of it in space as before, but now add a requirement for the magnitude of velocity to also be below some error.

VI. Lessons Learned

The clear lesson that vicon data reveals is that odometry readings do not agree with real positions. In testing the vehicle performed qualitatively better. So poor odometry could be due to the environment, or some hardware issue. The root cause of this should be investigated.

In terms of more directly actionable lessons, these results show a velocity requirement would be beneficial for more straight-line maneuvers between nodes, and there is reason to tune X-Y gains for overshoot.

VII. Conclusion

Despite a poor showing at demonstrations, the data from those tests is valuable at exposing issues that are not due to the trajectory controller, and illustrating room for improvement with regard to the trajectory controller itself.

References

[1] Sebastian OH Madgwick, Andrew JL Harrison, and Ravi Vaidyanathan. "Estimation of IMU and MARG orientation using a gradient descent algorithm." 2011 IEEE international conference on rehabilitation robotics. IEEE, 2011.

[2] Edgar Kraft. "A Quaternion-based Unscented Kalman Filter for Orientation Tracking." Sixth International Conference of Information Fusion. IEEE, 2003.

[3] https://prgaero.github.io/2019/proj/p1a/report

[4] *https://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf*

[5] *http://www.stengel.mycpanel.princeton.edu/Quaternions.pdf*

VIII. FIGURES

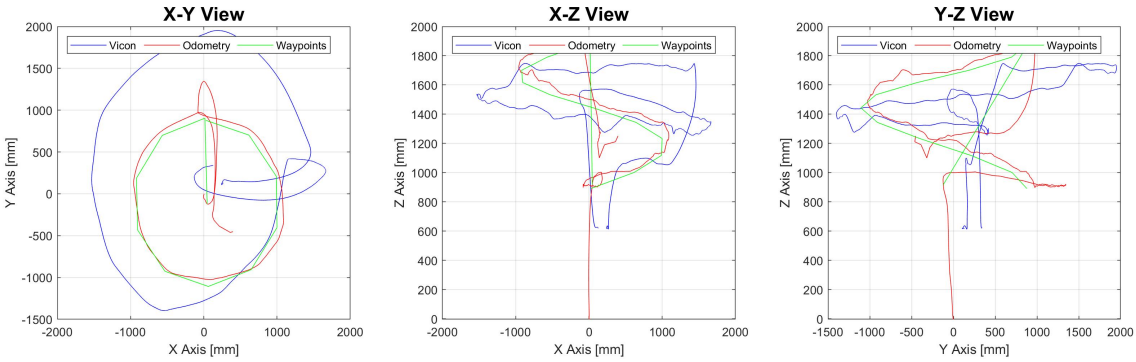

Figure 4. Trajectory 1: Helix Trajectory

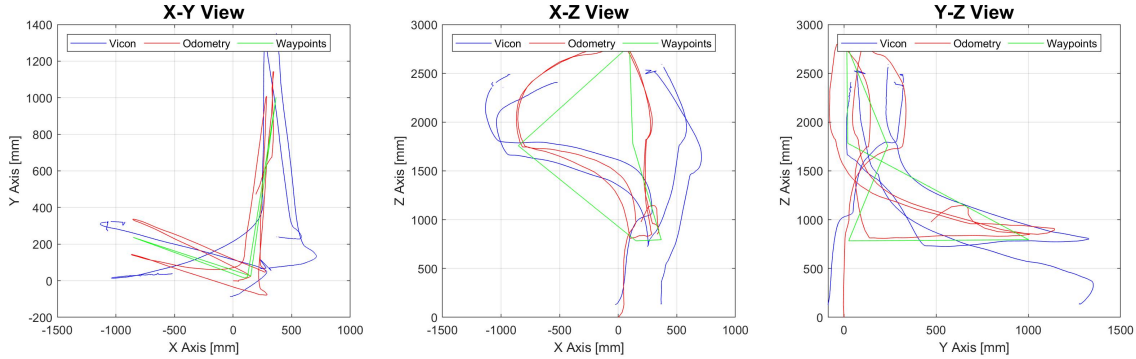

Figure 5. Trajectory 2: Diamond Trajectory

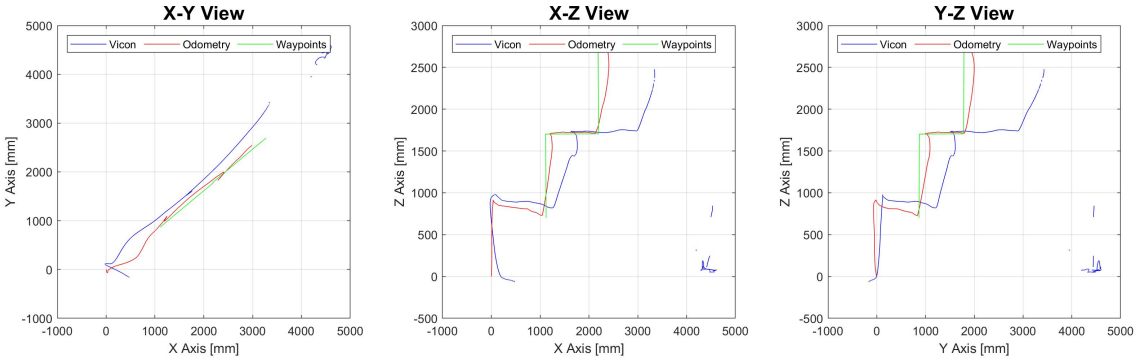

Figure 6. Trajectory 3: Staircase Trajectory