Final Report

For individual project

Jonas Brami, Brett Stephens

31 August, 2022

Contents

1. Introduction

The problem we aimed to solve during this project is the placement of a sensor on a specific target point on a surface using a fixed manipulator arm mounted on the top of an unmanned quadcopter.

A quacopter is an underactuacted helicopter with four rotors. A robot is said to be underactuated when arbitrary configurations cannot be realized.
In the last ten years, research about UAV controls accelerated drastically with many applications in the civilian industry in a variety of areas.
Monitoring and sensing tasks are traditionally operated by human but can be complicated (require expert, highly trained technicians), expensive, and dangerous for the human operator (when the point of interest is hard to reach, the human manipulator may need special training and equipement to reach the point of interest). For example, big structures like bridges need to undergo regular inspections to ensure there are no cracks or other signs of structural fatigue.
Using UAVs for such tasks would not only reduce cost of maintance of those structures, but also increase the security of both the human manipulator and the civilians using it.
Sensor placement on surfaces using UAVs is an active field of research and we will propose a solution based on velocity fields.

In this project we present an application of the Passive Velocity Field controller (PVFC) using Velocity field derived from the gradient of a potential or a shaping function; integrated with a depth camera for active obstacle avoidance and a grasping arm for sensor placement. We also provide an analysis of the controller behavior under multiple types of fields and controller parameters.

This project was done in 2 steps: The first one was the implementation of a Passive Velocity Field controller with simple quadcopter dynamics and physics on Python. The simplicity of this simulation has allowed us to easily debug the controller, verify that our implementation is behaving as expected, and get a high level understanding of the parameters of this controller. The second step was implementing the full Sensor Placement pipeline using ROS/Pixhawk 4/Gazebo. Robot Operating System is a platform to build robot application where task specific nodes interact together based on subscribing and advertising to topics. Gazebo is a simulator providing real world physics, a variety of plugins that we used to implement a grasping arm to simulate a Sensor Placement task. Pixhawk 4 is an open source autopilot sotfware integrated with ROS to provide seamless communication between our ROS controller and the quadcopter actuators. Those implementations will be further detailed in section 4.

This work would not have been possible without the initial implementations and huge help of my Project supervisor Brett Stephens !

2. Background and Literature Review

2.1. Placing sensors using UAVs

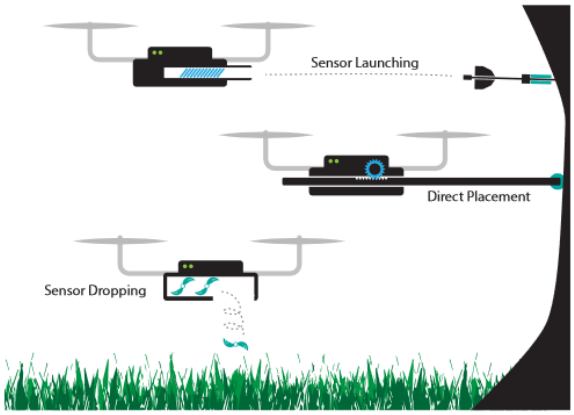
Diverse ways of placing sensors using UAVs have been explored in the past, including but not limited to:

- Direct Placement: Using a fixed arm manipulator on an UAV, we use the thrust of the UAV to provide enough pressure on the tip of the arm to place the sensor on the target point.
- Sensor Launching [2]: Using the energy stored in a spring, the UAV ejects the sensor at the desired velocity to reach and attach to the target (Unmanned Aerial Sensor Placement for Cluttered Environments). This strategy is very useful when it is not physically possible for a mounted arm to reach the target, however, it suffers from small payload capacity.
- Drop from flight: We simply drop the sensor above the target point. When target accuracy is not a priority and we are aiming at a non-vertical surface and there is no occlusion above the target, this sensor placement strategy is the most effective.

The characteristics of each one of mentioned methods are shown in figure 1. We decided to go through with the Direct Placement strategy because despite its simplicity, it provides good accuracy and is able to place a large variety of payloads.

2.2. Current State of the art for UAM

There exists 2 main approaches for controls of unmanned aerial manipulators: The first one is the centralized approach where we consider the manipulator and the UAV as a whole; whereas in the decentralized approach, the manipulator controls and UAV control are independant problems. In the case of the Sensor placement with a quadcopter, we use the centralized approach because the arm has no degree of freedom and the force exerted by the tip is coming from the thrust of the UAV. The centralized approach is often built on top of a model-based full state control loop optimized with LQR



	Direct [7]	Drop [2]	Launch [i]
Accuracy	±0.025 m	±4 m	±0.1 m
Safety distance	0 m	>10 m	4 m
Payload	1.85 kg	10 kg [ii]	0.65 kg
Sensor number	single	multiple	expandable

Figure 1: Different types of sensor placement from [2]

(Linear–quadratic regulator) around some desired state. In [6], the author present the current state of the research for UAMs. An UAM can be divided into 4 elements: the UAV floating base (in our case, a quadcopter), the robotic arm, a sensor/gripper attached to the end of the arm (in our case, a sensor will be attached to the end of the arm), diverse sensors on UAV to handle perception (the depth camera). While the state of the art approaches for UAV controllers are based on minimizing a state trajectory error, the planning strategy we will study next is based on velocity fields.

2.3. Velocity field path-planning for single and multiple unmanned aerial vehicles

In [2], the author presents a path-planning technique based on velocity fields generated from potentials solution of Laplace's equation. Two different types of solution to potential V for the Laplace

$$\nabla^2 V = 0$$
$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0 \tag{1}$$

equation are presented in this paper: Type 1 are irrotational solutions to generate sink and source fields and Type 2 solutions are used to build solenoidal fields.

$$V_1 = Q_1 \ln\Big((x_1 - \tilde{x_1})^2 + (x_2 - \tilde{x_2})^2\Big) \tag{2}$$

$$V_2 = Q_2 \arctan\left(\frac{(x_2 - \tilde{x_2})}{(x_1 - \tilde{x_1})}\right) \tag{3}$$

where (x_1, x_2) is the position of the UAV, $(\tilde{x_1}, \tilde{x_2})$ is the position of the obstacle; V_1 and V_2 are respectively type 1 solution (source field) and type 2 solution (vortex field).
The author justifies the use of Laplace solution for building the velocity field for multiple reasons:

- The use of Laplace solution for potential guarantees the uniqueness of the minimum in the field. Specifically, the use of vortex function built from shaping function to circle around obstacle will ensure that only the goal point will be a minimum of the field and that the UAV will not get stuck at some local minimum. As the author states, we can do an analogy to a famous strategy to find the exit of a maze: by keeping a hand on a wall of the maze and walking while always touching the wall, we are ensured to find the end of the maze. This is far from being an optimal solution, however, it can guarantee that the goal will be reached. As a result, those solenoidal fields based on vortex function also provide active collision avoidance.

- Scalar shaping functions are at the base of these methodology because by crafting them to match the shape of the obstacles, we are able to generate corresponding vortex or repulsive functions for obstacles of any shape. Since those functions are defined for each obstacle, it would be easy to reevaluate the field after addition or removal of an obstacle.

- Finally, irrotational solutions of the Laplace equation allow us to enforce an exclusion radius around obstacles (source field) and to direct the UAV in direction of the target point (sink field). The exclusion radius is encoded using the amplitude Q_1 of the irrotational field.

We can leverage these both types of potentials to derive a velocity field that will guide the UAV to the contact point without colliding with the surface. For example, we could define the exclusion radius to be the distance between the centre of mass (CoM) of the quadcopter and its most distant part on the quadcopter.

We will present the field we previously described with illustrations. The field in figure 2 was drawn by computing the gradient of V_1 with $(\tilde{x_1}, \tilde{x_2}) = (0, 0)$.

The field in figure 3 is a sink field at $(0, 1)$, it is similar to the source field in figure 2 but with opposite sign.

The field in figure 4 has been generated by computing the gradient of the shaping function of a

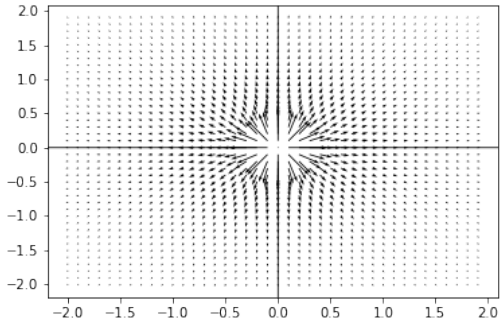


Figure 2: Simple Type 1 irrotational source

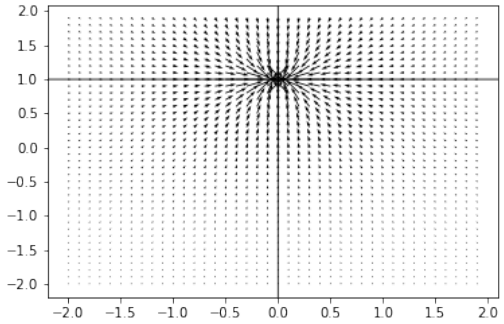


Figure 3: Simple Type 1 irrotational sink

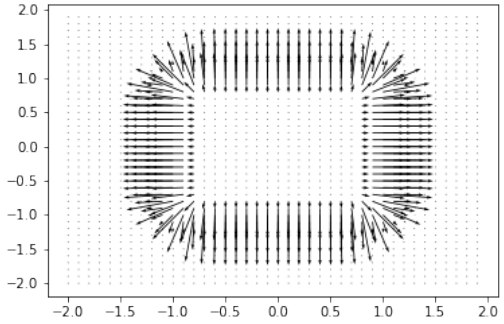


Figure 4: irrotational field from shaping function

superquadratic:

$$H = (x_1 - \tilde{x_1})^n + (x_2 - \tilde{x_2})^n \tag{4}$$

$$F = \frac{1}{1 + (\frac{1}{L} H^{\frac{1}{n}})^m} \tag{5}$$

As stated in [5], when $m \gg 1$, the edge of the shaping function gets thinner. Higher values of n result in more rectangular shapes whereas $n = 2$ describes the shaping function of an infinite cylinder.

Both these fields are type 1 irrotational solutions of the Laplace equation.

By adding the irrotational sink from figure 3 and the irrotational field from shaping function in figure 4 we obtain a good representation in figure 5 of what the velocity field will look like when close to the target point.

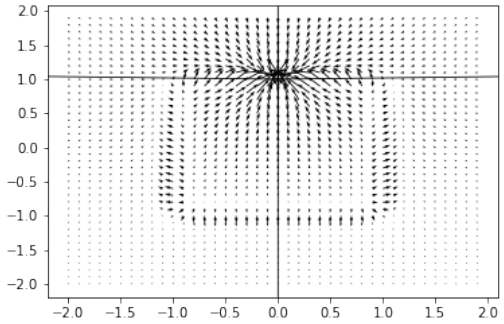


Figure 5: irrotational field from shaping function with sink

The shaping function is mainly used to generate a vortex field around an obstacle, as explained in the paper. It is given by:

$$v_2 = -\frac{\partial F}{\partial x_2} e_1 + \frac{\partial F}{\partial x_1} e_2 \tag{6}$$

Here, e_1 and e_2 are an orthonormal basis.
We can see in figure 6 an example of vortex field around a super quadriatic.

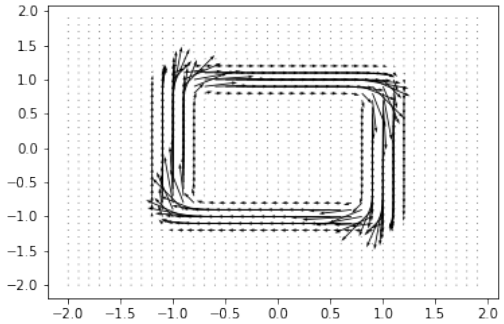


Figure 6: Solenoidale field from shaping function

2.4. QUAD PD CONTROL example

3. Design and Theory

3.1. Designed velocity Field for planning task

A velocity field is a function taking as input a state, q and returns a 3-dimensional desired velocity vector (u, v, w).
It describes a path (a function of position) as opposed to a trajectory (function of position and time).

3.1.1. Sink field for reaching the target

In [5], the author chose to apply a logarithmic function on the distance between the current position and the goal position as a potential for the sink field. However, we observed that using such a potential does not allow the robot to safely approach the goal point since when the distance approaches 0, the logarithmic function diverges. As a result, we decided to simply use the distance as a potential function. This will allow us to smoothly decrease the velocity of the UAV when approaching the desired goal point.

3.1.2. Obstacle repulsive field

In our current solution, because of the simplicity of the environement and the convexity of the obstacles, we are only using the type 1 solution described in [5], we only have a repulsive field normal to the surface of the obstacle.

3.1.3. Spherical field to maintain desired force

When contact has been made, we suppose that the surface static friction coefficient is high enough to maintain the contact. Since the arm has a fixed size and does not move, this section will describe a velocity field on the surface of a sphere around the target point with a radius defined by the distance between the CoM of the quadcopter and the tip of the arm.

First we need to compute the feasible position of the CoM to apply the desired force. We know that this position is unique because there is only one vertically stable pitch for a given desired force amplitude. The quadcopter needs to pitch to have a forward velocity because the quadcopter is underactuacted.

We can generate the velocity field on the surface of the sphere to point on the tangent direction of the sphere in the direction of the stable pitch position. This field will have an amplitude proportional to the distance from this point. The planning strategy we described would also allow us to easily define a desirable range for the yaw angle depending on the type of sensor and on the friction coefficient of the target surface. Finally, we use a Passive Velocity Field Controller [4] to follow this field to minimize the loss of kinetic energy to the environment when interacting with the surface. Now we are going to describe how the spherical field is computed. Let us recall that the cartesian to spherical coordinate change of variable is defined as

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \arccos\left(\frac{z}{r}\right)$$

$$\phi = \begin{cases} \arctan\left(\frac{y}{x}\right), & \text{if } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi, & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi, & \text{if } x < 0 \text{ and } y < 0, \\ \pi/2, & \text{if } x = 0 \text{ and } y > 0, \\ -\pi/2, & \text{if } x = 0 \text{ and } y < 0, \\ \text{not defined}, & \text{if } x = 0 \text{ and } y = 0 \end{cases} \tag{7}$$

The jacobian defining the curvature of the sphere at each point for this transformation is

$$J = \frac{\partial(x, y, z)}{\partial(r, \phi, \theta)}$$

$$J = \begin{bmatrix} \sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) & -\sin(\phi) \\ \sin(\theta)\sin(\phi) & \cos(\theta)\sin(\phi) & \cos(\phi) \\ \cos(\theta) & -\sin(\theta) & 0 \end{bmatrix} \tag{8}$$

Figure 7 represents an example of such a field where the red point is the optimal position to apply the pressure, the blue arrows are the velocity field vectors and the center of the sphere is the point where the force is applied

3.1.4. Circular field for countour following

This field is based on the methodology explained in [1]. We used it to build intuition, debug and analyze the controller. The author explains how to encode a countour $\mathcal{C}$ into a desired velocity field v_d. The field is given by:

$$v_d(p) = l(\alpha \hat{t} + \|Q - p\|\hat{n}) \tag{9}$$

- $p \in \mathbb{R}_3$ is the current position
- Q is the closest point from p on $\mathcal{C}$
- $l, \alpha \in \mathbb{R}_{\geq 0}$ are scaling constants
- $\hat{t}$ is a unitary tangent vector to $\mathcal{C}$ at Q
- $\hat{n}$ is a unitary normal vector to $\mathcal{C}$ going from p to Q

From there it is straightforward how to implement a circular velocity field with a radius of 1 and centered at the origin in the following way:

$$Q = \frac{p}{\sqrt{p_x^2 + p_y^2}} \tag{10}$$

$$\theta = \arctan(Q_y, Q_x) \tag{11}$$

$$\hat{t} = (-\sin\theta, \cos\theta, 0) \tag{12}$$

$$\hat{n} = \frac{Q - p}{\|Q - p\|} \tag{13}$$

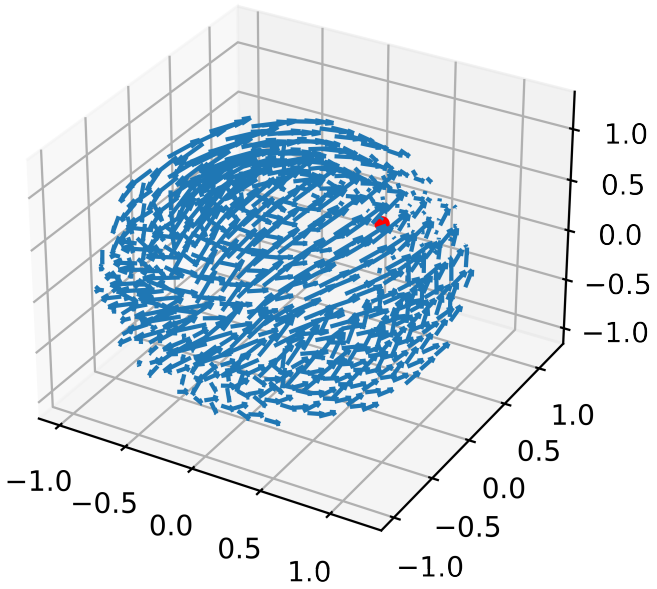


Figure 7: Spherical contact field

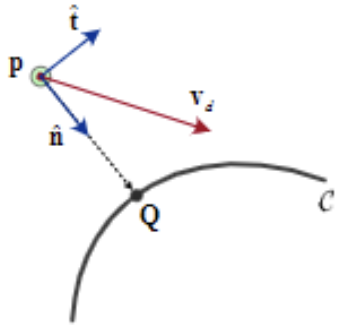


Figure 8: field construction from [1]

3.2. Quadcopter modeling

3.2.1. Platform

Similarly to the platform used in "An Aerial Parallel Manipulator with Shared Compliance", our system is composed of two subsytems: The underactuated quadrotor base (for which we have software in the loop integration given by PX4) and the fixed manipulator arm above the center of mass of the quad. This system is underactued because in the base frame of this quad, the only thrust direction for the quadcopter is in the z axis in the frame of the base of the quad, the rotors cannot move relative to the base of the quad. Therefore, the IRIS model control inputs are a quarternion vector and a thrust amplitude. However, the output of the passive controller for the non-augmented translational domain is a thrust vector $\Lambda \in \mathbb{R}_3$, therefore we will need to convert this desired thrust to the desired format. In addition to the fixed manipulator arm, a depth camera is mounted on the quad base. We use the depth camera model provided by Pixhawk 4 with IRIS that acts like an intel realsense r200. We use the transform provided in the PX4 obstacle avoidance package to transform from the frame of the camera to the frame of the quadcopter base, and we use the PX4 odometry information to transform from the base to the ground gazebo frame.

3.2.2. Translational dynamics

The dynamics that we take into account in our passive control strategy are only in the translational domain. The passive velocity field controller output is a thurst vector and PX4 translates this desired thurst into actual angular velocities for each one of the rotors $\Omega = (\bar{\omega_1}, \bar{\omega_2}, \bar{\omega_3}, \bar{\omega_4})$. PVFC only acts on the translational level and not at the rotational level and has no knowledge or control about the speed of thoses rotors and therefore the passivity of this controller is only at the translational level. This quadcopter is an underactuacted sys-

tem because for a given translational velocity, only a subset of attitudes are feasible. As a result, it is not possible to control all 6 degrees of freedom independantly (3 translational, 3 rotational).

3.3. Passive Velocity Field Control of Mechanical Manipulators

In [4], the author explains the advantages of encoding a contour following task using velocity fields. They later present the passive velocity field controller whose objective is to "maintain an energetically passive relationship between the manipulator under closed loop control and its physical environment, while causing the manipulator to perform the desired task." We will explain each one of those arguments and relate them to our sensor placement task.

3.3.1. Velocity fields for planning

The classical approach to do planning is to encode the task into a timed trajectory $Q : \mathbb{R}_{\geq 0} \to G$ where G is the n-dimensional configuration manifold for the manipulator and uses a controller to minimize the state trajectory error. Using this strategy, the objective of the controller is to minimize the deviation between $q(t)$ and $Q(t)$ where $q : \mathbb{R}_{\geq 0} \to G$ is the actual coordinate representation of the manipulator. This strategy could be fine if the manipulator was able to never deviate from the timed trajectory defined by Q. However, if deviation occurs for example, when external forces are applied to the robot, a side effect of this minimization strategy called radial reduction can occur.

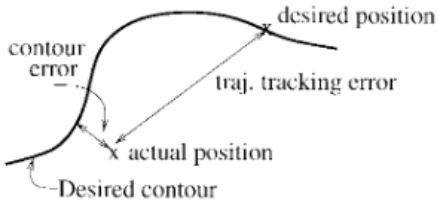


Figure 9: radial reduction from [4]

The author argues that if following the path of the desired trajectory is more important than the timing in which the manipulator follows this trajectory, a strategy based on velocity field is more appropriate because the velocity of the manipulator will only depend on its current position and will be time invariant. For the sensor placement task, using a timed trajectory strategy with deviation minimization to reach the contact point may lead the UAV to collide with an obstacle. With our strategy based on potential velocity field, the UAV will not try to shortcut the desired path in case of deviation. The author defines an α error such that:

$$e_\alpha = \dot{q} - \alpha V \tag{14}$$

One of the most important priorities of PVFC is that when no external forces are applied on the robot, there is a positive α such that

$$\lim_{t \to \infty} e_\alpha = 0 \tag{15}$$

In other words, PVFC does not seek to exactly match the desired velocity field but just an α scaled version of it. The α we are going to converge to is a function of the energy in the augmented system and the energy in the desired velocity field.

3.3.2. Passivity

The controller presented in this paper is based on energy control theory and allows energy transfer between the environement and the augmented system. Passivity allows dissipation from the energy input to the system into the augmented system.
To present this concept, the author first defines the notion of a passive dynamic system:
A dynamic system with input $u \in U$ and output $y \in Y$ is passive with respect to the supply rate $s : U \times Y \to \mathbb{R}$, if for any $u : \mathbb{R}_{\geq 0} \to U$ and for any $t \geq 0$ the following relation is satisfied:

$$\exists\, c \in \mathbb{R}$$
$$\int_0^t s(u(\tau), y(\tau)) d\tau \geq -c^2 \tag{16}$$

Let us remember that the work W and power P of a force F on a point mass object with velocity v are defined as

$$P = F \cdot v \tag{17}$$
$$W = \int_0^t P dt \tag{18}$$

Therefore the supply rate $s(\tau_{tot}, \dot{q}) = \tau_{tot}^T \dot{q}$ can be seen as the total mechanical power input. Here the input τ_{tot} is the total force exerted on the manipulator and the output $\dot{q}$ is the velocity of the manipulator.

When considering a feedback system interacting with the environement shown in figure 10, we can decompose $\tau_{tot} = \tau_e + \tau$ (where τ and τ_e are respectively the forces generated by the actuators and the external forces, for example the contact force when touching a surface), we can derive the power generated by external forces as $s(\tau_e \dot{q}) = \tau_e^T \dot{q}$. As explained in the paper, a system defined by this supply rate is not passive because obstacles may bring the manipulator to a complete stop for an unbounded amount of time and this loss

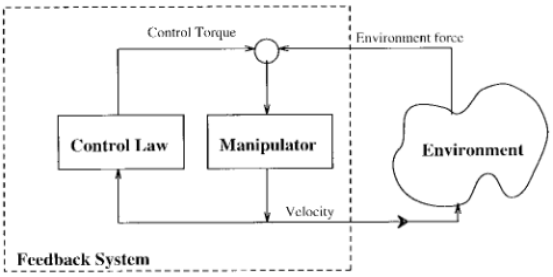


Figure 10: PVFC loop [4]

of kinetic energy cannot be bounded. As a result, the passivity relation

$$\int_0^t \tau_e^T \dot{q} d\tau \geq -c^2 \tag{19}$$

is not valid here because the l.h.s represents the total amount of energy lost to the environment and, as stated before, it cannot be bounded. This issue motivates the introduction of an augmented system: a fictitious flywheel is added to the system that acts as an energy storage element. The dimensionality of the manifold is then increased by one to include the state of the flywheel and this augmented state will be noted as

$$\bar{q} = [q, q_{\text{fly}}] \tag{20}$$
$$\bar{V} = [V, V_{\text{fly}}] \tag{21}$$

Using such passitive and dissipative systems enables safe interaction between the robot and its environement since the total kinetical energy of the system is bounded by the system initial kinetic energy and energy input from the environement.

3.3.3. Augmented dynamics

Let us now describe the dynamics of the augmented system

$$\bar{M}(\bar{q})\ddot{\bar{q}} + \bar{C}(\bar{q}, \dot{\bar{q}})\dot{\bar{q}} = \bar{\tau} + \bar{\tau_e} \tag{22}$$

The mass matrix of the augmented system is defined such that:

$$\bar{M}(q) = \begin{bmatrix} M(q) & 0 \\ 0 & m_{\text{fly}} \end{bmatrix} \tag{23}$$

$$M(q) = \begin{bmatrix} m_b & 0 & 0 \\ 0 & m_b & 0 \\ 0 & 0 & m_b \end{bmatrix} \tag{24}$$

The Coriolis Matrix of the augmented system is defined such that:

$$\bar{C}(\bar{q}, \dot{\bar{q}}) = \begin{bmatrix} C(q, \dot{q}) & 0 \\ 0 & 0 \end{bmatrix} \tag{25}$$

C is built using the Levi–Civita connection described in [3].

3.3.4. Augmented velocity field

Now let us define the augmented field for the flywheel. The motivation for introducing an augmented system is to allow energy transfer between the quadcopter actuators and the flywheel such that the total kinetic energy in the system remains constant. This property is enforced in the definition of the desired velocity field.

First let us define $\bar{E}$ to be the total desired energy of the augmented system following the desired velocity field. $\bar{E}$ can be written as a sum of the flywheel desired kinetic energy K_{fly_d} and the quad desired kinetic energy K_{quad_d}:

$$\bar{E} = K_{\text{fly}_d} + K_{quad_d} \tag{26}$$

Given a desired velocity field V we have:

$$K_{quad_d}(q) = V(q)^T \frac{1}{2} M(q) V(q) \tag{27}$$

Since

$$K_{\text{fly}_d} = \frac{1}{2} m_{\text{fly}} V_{\text{fly}}^2 \tag{28}$$

We can derive V_{fly} in function of $K_{quad_d}(q)$ and $\bar{E}$:

$$V_{\text{fly}}(q) = \sqrt{\frac{2}{m_{\text{fly}}}(\bar{E} - K_{quad_d}(q))} \tag{29}$$

3.3.5. PVFC Control law

To define the PVFC control law, the author defines the following quantities

$$\bar{p}(\bar{q}, \dot{\bar{q}}) = \bar{M}(\bar{q})\dot{\bar{q}} \tag{30}$$
$$\bar{P}(\bar{q}) = \bar{M}(\bar{q})\bar{V}(\bar{q}) \tag{31}$$
$$\bar{w}(\bar{q}, \dot{\bar{q}}) = \bar{M}(\bar{q})\dot{\bar{V}} + \bar{C}(\bar{q}, \dot{\bar{q}}) \tag{32}$$

The equation 30 can be seen as the actual momentum of the augmented system, the equation 31 can be seen as the desired momentum of the augmented system and 32 is the desired dynamics (desired force applied on) of the augmented system.

From there the author derived the two terms of the coupling control law:

$$\bar{\tau_c} = \frac{1}{2\bar{E}}(\bar{w}\bar{P}^T - \bar{P}\bar{w}^T)\dot{\bar{q}} \tag{33}$$
$$\bar{\tau_f} = \gamma(\bar{P}\bar{p}^T - \bar{p}\bar{P}^T)\dot{\bar{q}} \tag{34}$$
$$\tag{35}$$

The anti-symmetric structure of the matrices multiplying $\dot{\bar{q}}$ is key to proving that the derivative of the kinetic energy in the augmented system is:

$$\frac{d}{dt}k(\bar{q}, \dot{\bar{q}}) = \tau_e^T(t)\dot{q}(t) \tag{36}$$

This is done by introducing the concepts of compatibility between a metric defining an inner product and an affine connection. It is shown in [3] that the compatibility between the Levi-Civita connection with the metric defined by M has a direct link between anti-symmetricity of the matrices in the control coupling law, passivity and energy conservation.

From equations 36 and 16, the passivity of the system with respect to external forces is straightforward.

3.4. Compensation terms

When using PVFC with the quadcopter dynamics, it is important to model the effect of drag and gravity on the robot to be able to compensate for them. If those forces are not taken into account, the robot total mechanical energy will decrease until it arrives to a complete stop. Therefore, to ensure the robot will continue to follow the desired velocity field, we have to introduce compensation terms for gravity and air drag. For simplicity, we chose to model drag as a linear function of the velocity. This worked for our proof of concept but this method shows its limits when running the simulation for a greater amount of time because not only drag is not exactly proportional to the speed, but we have no straightforward way of estimating those drag coefficients. Setting them too high will make our control ouput diverge but setting it too low will make to quad stop following the desired velocity field.

4. Implementation

As was stated in the introduction, we have two main implementations:

4.1. Simplistic Python implementation

The Python implementation uses simplistic quadcopter dynamics only defined by:

- Quad mass: m_b
- Moments of inertia: I_{xx}, I_{yy}, I_{zz}
- drag coefficient: A_x, A_y, A_z
- PD attitude gains: K_{ϕ_p}, K_{ϕ_d}, K_{ψ_p}, K_{ψ_d}, K_{θ_p}, K_{θ_d}. ϕ is the roll angle, θ is the pitch angle and ψ is the yaw angle.

In this subsection, we will consider $q \in \mathbb{R}_6$ to be the rotational and translational states: $q(t) = (x(t), y(t), z(t), \phi(t), \theta(t), \psi(t))$ and $\dot{q}(t) = \frac{dq}{dt} = (\dot{x}(t), \dot{y}(t), \dot{z}(t), \dot{\phi}(t), \dot{\theta}(t), \dot{\psi}(t))$

It should be noted that the drag coefficients here are used both by the PVFC controller and the dynamics of our system. In opposition to the Gazebo implementation, the drag experienced by the quad in the python implementation is also defined by those drag coefficients. When running in Gazebo, the IRIS model has a real world drag that we do not control.

We use ODEint to evolve the state $(q, \dot{q}, \ddot{q})$ of the system by doing to following on each time step:

- First we compute the current Mass Matrix M, Coriolis Matrix C, Control Matrix B, gravity Matrix G. Since the mass of the system is constant, the translational parts of M and G are constants. Therefore, the only changes we have to take into account are for C and B.
- Then we pass to the PVFC controller those dynamics together with the desired velocity field and compute a desired thurst vector.
- Given a desired cartesian thrust vector $\tau_{desired} \in \mathbb{R}_3$, we can decompose it into euler coordinate $(\alpha, \tau_{\phi_{desired}}, \tau_{\theta_{desired}}, \tau_{\psi_{desired}})$.
- The quad uses a simple attitude PD to compute the final thrust $\tau(t) = (\tau_\phi(t), \tau_\theta(t), \tau_\psi(t))$ in euler coordinate :

$$\tau_\phi(t) = I_{xx}(K_{\phi_d}(\dot{\phi}_{desired} - \dot{\phi}(t)) + K_{\phi_p}(\phi_{desired} - \phi(t))) \tag{37}$$
$$\tau_\theta(t) = I_{yy}(K_{\theta_d}(\dot{\theta}_{desired} - \dot{\theta}(t)) + K_{\theta_p}(\theta_{desired} - \theta(t)))$$
$$\tau_\psi(t) = I_{zz}(K_{\psi_d}(\dot{\psi}_{desired} - \dot{\psi}(t)) + K_{\psi_p}(\psi_{desired} - \psi(t)))$$

- By multiplying the final euler thrust by the control matrix, we can obtain the new state of the quadcopter

It is important to notice that the big difference between the python implementation and the ROS/Gazebo/PX4 that we will later detail is that we do not explicitly use a PD attitude gains in the latter. The PX4 firmware of the IRIS model has its own low level attitude controller.

The desired velocity fields are computed using Sympy (A Python library for symbolic) for easy field and field gradient computation. Since most fields we used are derived from potential functions, we can easily obtain the field and field jacobian required by PVFC by symbolically differentiating their potentials or shaping functions.

4.2. Full Sensor Placement pipeline in ROS/Gazebo/Pixhawk 4

In this implementation, we are using a real world quadrotor called IRIS quad provided in the Pixhawk 4 SITL package. The dynamics are evolved

and computed by Gazebo. The implementation is divided in 2 ROS Nodes, 1 ROS-PCL Node, 1 Gazebo-ROS plugin, and 1 Pixhawk 4/MAVROS node.

4.2.1. Velocity Field ROS Node
This node subscribes to the mavros odometry topic, computes the velocity field at the current position and publishes the desired field and field gradient. We implemented all the velocity field using the symbolic math library SymEngine so that all gradient computations from potential or shaping function would be done automatically. In addition, this node is implementing a state machine such that in each state, a specific field is being used. Specifically, the first state gives a sink to get altitude, the second state is a sink field to approach the target, the third state represents the contact phase with the wall (the node subscribes to a topic called "grasping" to know if the grasping arm is currently activated). It starts when an engagement signal is received from the grasping arm and it finishes when a disengagment signal is received from the grasping arm. The last state is to get away from the wall and land, we use a sink field to implement it. We implemented an obstacle avoidance system for generic superquadratic obstacles, upon obstacle detection, we compute the avoidance field for this obstacle from the appropriate shaping function, an irrotational field is generated to avoid the obstacle. This node subscribes to an obstacle topic to know when new obstacles have been detected.

4.2.2. PVFC ROS Node
This node subscribes to the desired velocity field and to the MAVROS odometry and publishes a thurst vector on the MAVROS attitude topic computed by the PVFC controller. The output of the passive velocity field controller is a 4-dimensional vector containing the desired thrust to be applied in each direction (x, y, z, fly) including the thurst to be applied on the fictitious flywheel. However, the input control for the IRIS model needs a desired attitude and thurst amplitude. Therefore, after obtaining the desired (cartesian) thrust vector from PVFC, we convert it to desired euler angles with desired amplitude $(\alpha, roll, pitch, yaw)$. We can then convert this pose to quaternion and publish it to the attitude topic of the IRIS quad. It should be noted that the thrust amplitude expected by the IRIS attitude control is not in Newtons but is a scalar between 0 and 1. As a result, we had to experimentally calibrate a thrust slope and thrust intercept (affine transformation) to map the desired PVFC thurst amplitude to a scaled thurst amplitude.

4.2.3. PCL Object Segmentation Node
This node reuses the code of a PCL tutorial performing cylinder segmentation on point cloud data(cite). We added a ROS integration to the velocity field topic the position of the detected object and also we implemented the transforms from camera point of view to FCU (Flying control unit) frame, and from FCU frame to world frame. It works in the following way:

- Use a PCL passthrough filter to remove Nan datapoints and the scene background
- Use the PCL NormalEstimation class to estimate the normals of the cleaned pointcloud scene. This is done using KdTree based methods for increased robustness against noise in the measurements. KdTrees are data structures used in Computer Science that are very efficient for nearest neighbour search
- Perform plane model segmentation on the extracted normals using RANSAC and filter out the planar inliers
- After the last step, we should have a clean normals containing only cylinders normals, we can now use the SACSegmentationFromNormals class of PCL to extract the coefficient of the cylinder and its position in the frame of the camera
- If a cylinder was detected, we transform the position of the cylinder from the depth camera frame to the gazebo world frame and we publish it to the cylinder topic for the velocity field node to update the desired velocity field

4.2.4. Grasping arm Gazebo-ROS plugin
This node reuses the code of the vaccuum arm of Gazebo, but it was modified so that could be "attached" to the moving IRIS model and fixed to a wall in Gazebo.

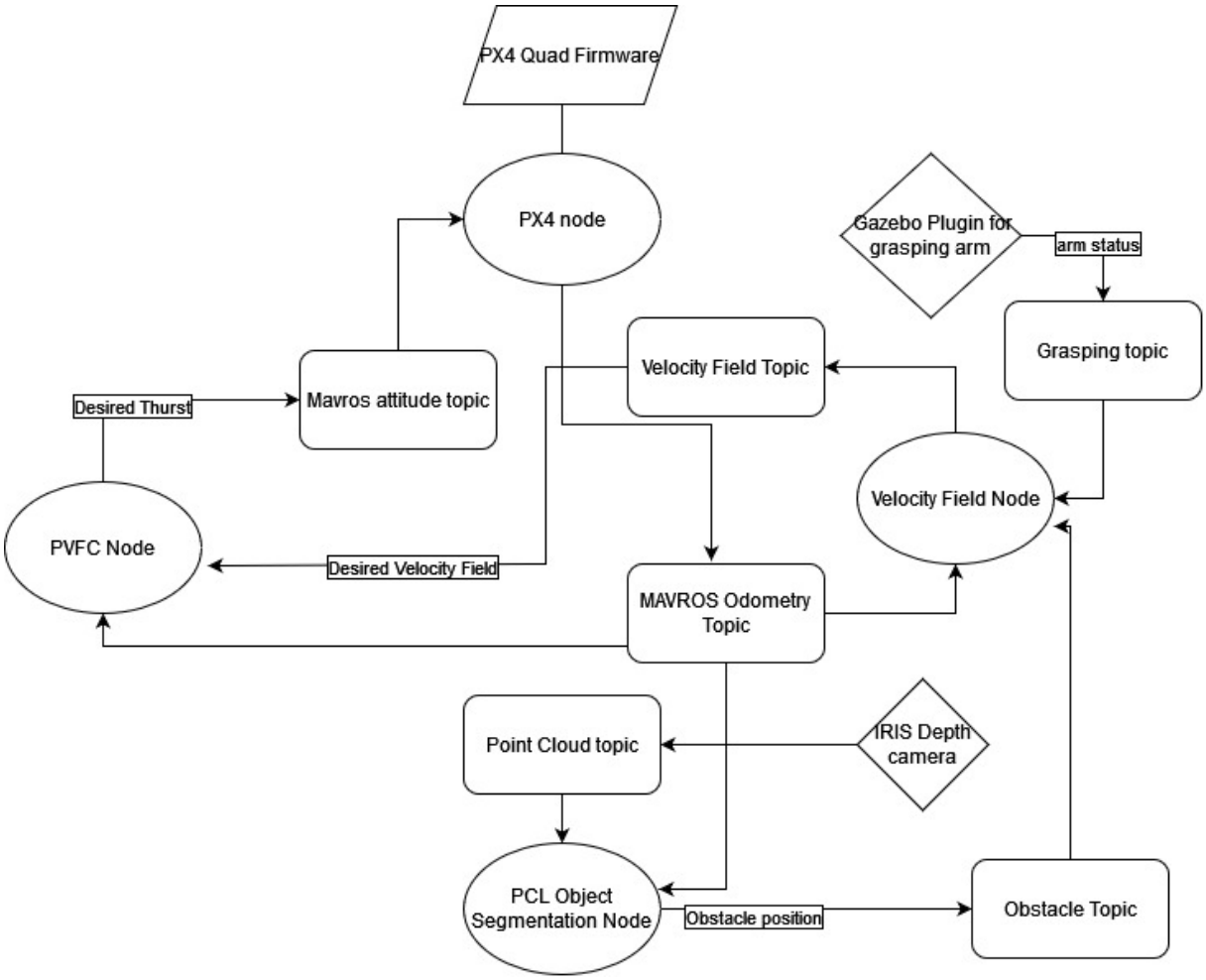


Figure 11: implementation diagram

5. Results, Evaluation and Experiments

In this section, we will discuss the results obtained in multiple conditions. Those data were recorded using ROSBag and were parsed using the ROSBag python module. We provide a jupyter notebook in the repository that facilitates drawing all the relevant graphs with a convenient way of storing all the parsed bags inside a pickle file.

5.1. Compensation for drag

In the first experiment, we will justify the need for a air drag compensation term. The velocity field we use in the circular velocity field based on [5].

5.1.1. Experience without drag compensation

On the both of figure 12, we can see the trajectory of the quadcopter, the left scatter plot of the trajectory is colored according to the translational velocity the quadcopter while the right scatter plot is colored according to the angular velocity of the fictitious flywheel. On both subplot, the quiver plot represents the desired velocity field We can see that when we do not introduce a drag force compensation, the quad translational velocity goes to zero. This is explained by the fact by the quadcopter thrust given by PVFC has a too low amplitude to compensate the drag force, the quad decelerate. The amplitude of the flywheel velocity is not affected by this drag because we defined a perfect no friction flywheel.
These subplots are useful to observe how energy is being transfered between the fictitious flywheel and the quadcopter. We can see that as expected from equation 29, the faster the quad is going the slower the fictitious flywheel is spinning.

Despite the total stop, we can see from the figure 13 that the field is accuratly followed

5.1.2. Experience with drag compensation

This experiment is the same as the last one except that we add to the desired pvfc thrust output the compensation term $\tau_{comp} = C \cdot \dot{q}$ where C is current translational coriolis matrix and $\dot{q}$ is the current translational velocity of the quadcopter. We can see in figure 14 that introducing this force

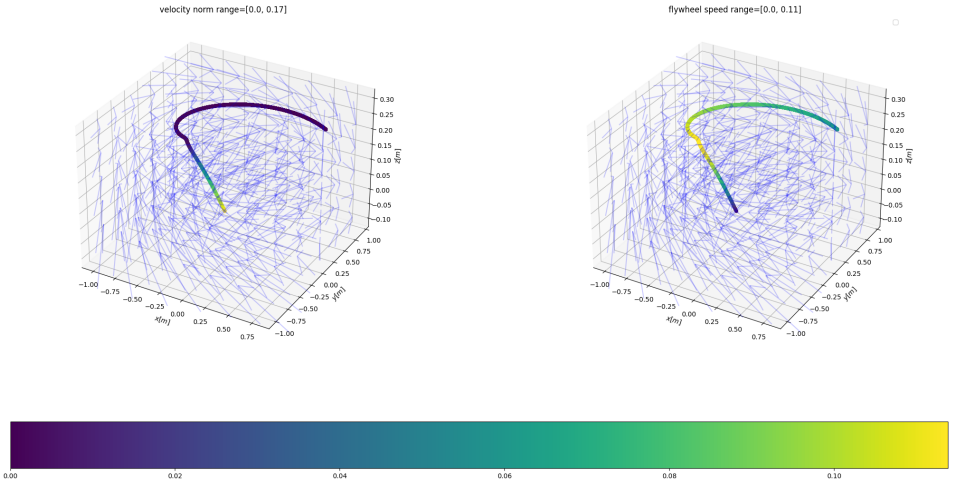


Figure 12: trajectory and mechanical energy without force compensation

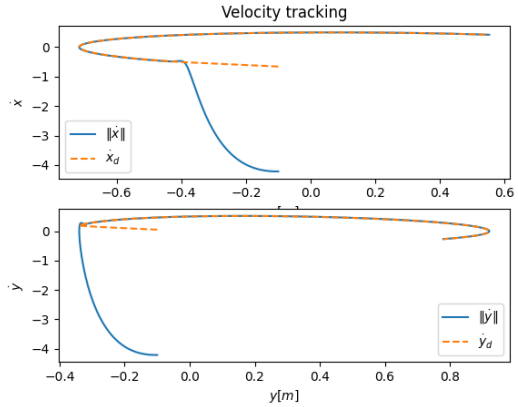


Figure 13: Trajectory tracking without force compensation

compensation allows the quad to continue moving despite drag forces. However, use such compensation slightly decreases the precision of the velocity field tracking (figure 15). It is not clear why this is happening.

5.2. No drag forces

In this second experiment, we remove the drag forces from the dynamics (we set the drag coefficients to 0). We can see in the figure 16 that when far away from the desired path, the translational speed of the quadcopter is high while the rotation speed of the flywheel is low. This is happening because of the way the veloicty field is constructed in equation 9, the magnitude of the tangential component is constant but the magnitude of the normal component is proportional to the distance between the quad and the closest point in the circle. We can observe how energy is being traded between the quad and the flywheel when the quad approaches the desired path. It should also be noted that the desired path is exactly followed in opposition to the last experiment with drag compensation (see figure 17) We explained in section 3 that PVFC has interesting properties regarding the velocity tracking error (eq 14). The author in [4] explains that When no external forces are applied on the quadcopter, α should approach $\beta^2 = \frac{\text{augmented mechanical energy}}{\text{augmented desired mechanical energy}}$.
In addition, when α approach β, the *alpha* error should approach 0.

By design, the augmented desired mechanical energy of the system is always equal to $\bar{E}$, and we compute the augmented mechanical energy such that:
augmented mechanical energy $= \frac{1}{2}m_b\|\dot{q}\| + \frac{1}{2}m_{fly}\|\dot{q}_{fly}\|$
We successfully reproduced the results of [4] for the β error on a circular velocity field. We can see in figure 18 that β converges and the β error converges to 0.

5.3. No drag forces with force wrench

The third experiment is done in 3 phases: In the first one we let the quad follow the circular field for 120 seconds, then we apply an external force on the quad in the $-\hat{y}$ direction for 10 seconds, and finaly we let the quad follow the desired field for 120 seconds. The passivity properties of PVFC are well shown in figure 19. We can see how the

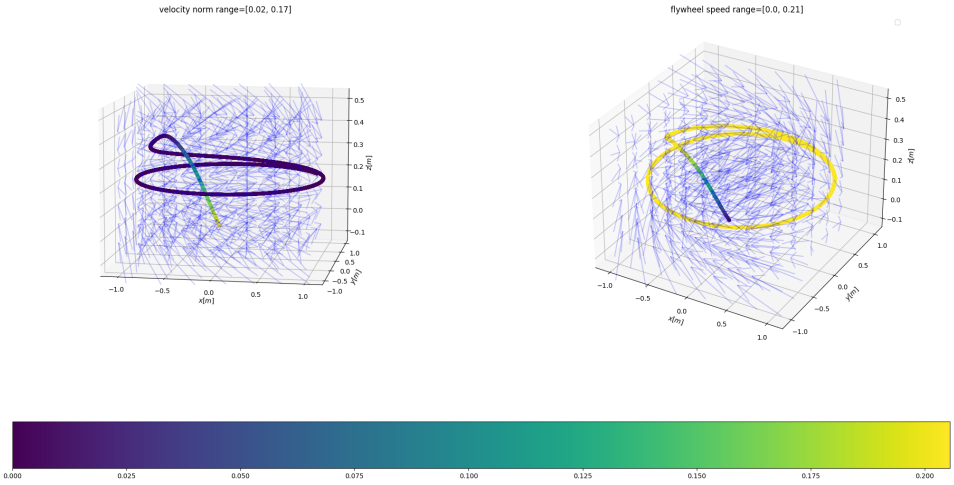


Figure 14: trajectory and mechanical energy with force compensation

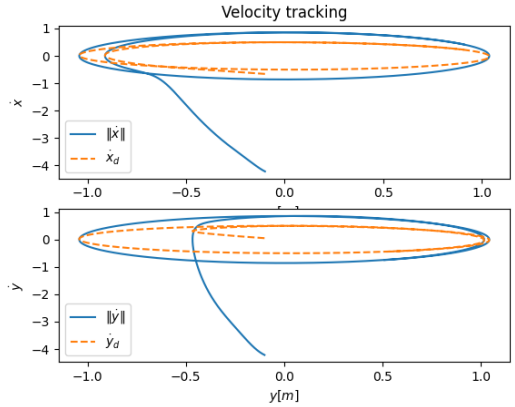


Figure 15: Trajectory tracking with force compensation

quadcopter is trading energy with the environement and dissipating the work of the wrench inside the augmented system. This experiment also exemplify well the advantage of using velocity field instead of a traditional trajectory following based controller: The radial reduction that could have occured in figure 20 could have been a big issue in a task such as countour following where following the countour is more important than following a trajectory. In the contrary, figure 20 shows that when external forces are applied, we are able to go back to the desired circular path without radial reduction. In addition, we can observe what happens to the β error when wrench is applied: We can see in figure 21 that the β error converges to 0 before a wrench is applied, then during the second phase, we can see a spike for both β and β error when a wrench is applied. This is expected since the wrench increases mechanical energy in the system. We can see how PVFC dissipates the external force and makes the β error converge to 0 when no external force is applied in the last phase.

Now we are going to show experiments done on the Gazebo simulation

5.4. Gusty world

5.5. Windy world

5.6. gamma analysis

5.7. Ebar analysis

- Calibrate thrust
- introduction of compensation terms for friction
- finetuning (Ax Ay Az). too high cause speed to diverge, too low the quad lose all its energy and stop.
- beta error
- presents experiences in different worlds
- using wind world helped lead the way to understand the need for compensation
- Analyze Ebar, Gamma

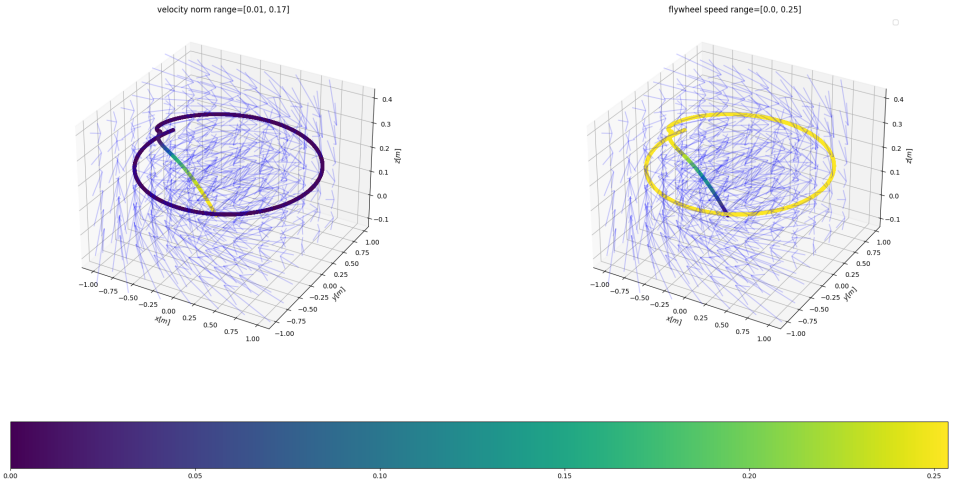


Figure 16: trajectory and mechanical energy without drag

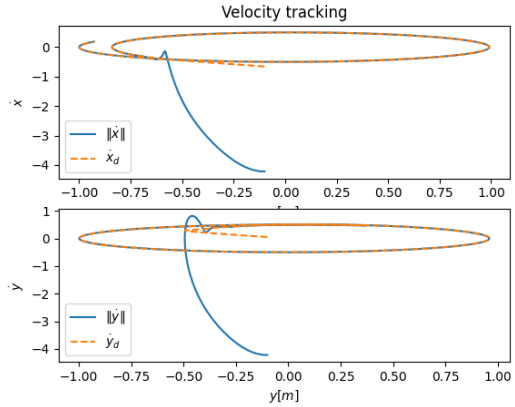


Figure 17: Trajectory tracking without drag

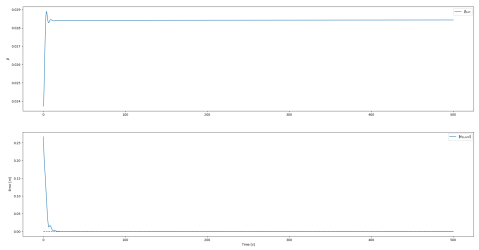


Figure 18: Beta error without drag

- Smart sampling the field to find the best Ebar
- More obstacle types
- desired yaw velocity field
- switch to circular field when stuck

6. Conclusion, Future Work

- Evolutionary algorithm to select the optimal gains and Ebar
- Estimate the friction coefficient

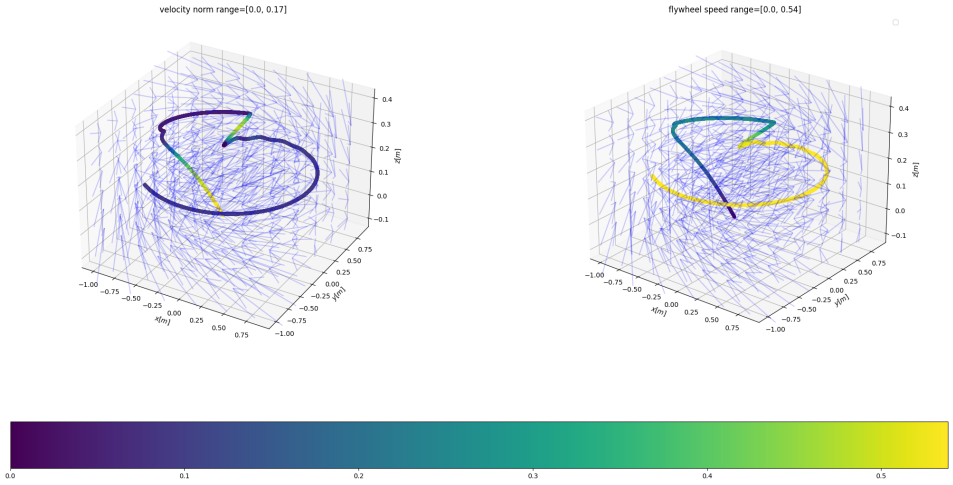


Figure 19: trajectory and mechanical energy without drag and with wrench

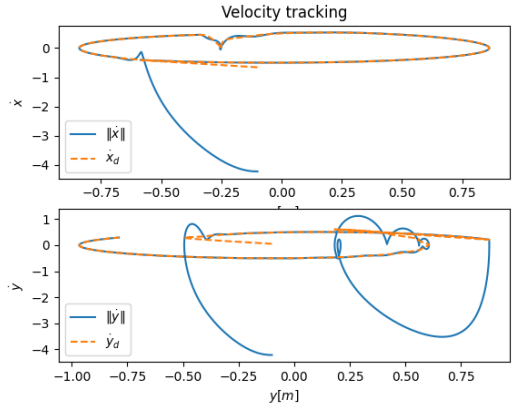


Figure 20: Trajectory tracking without drag with wrench

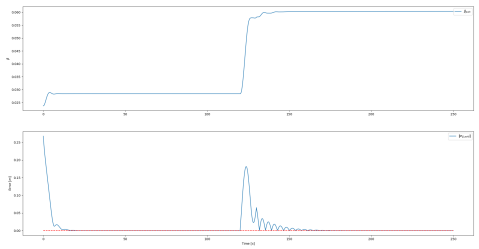


Figure 21: Beta error without drag with wrench

References

[1]H. J. Asl and T. Narikiyo, ≪An assistive control strategy for rehabilitation robots using velocity field and force field≫, in 2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR) (IEEE, 2019), pp. 790–795.

[2]A. Farinha, R. Zufferey, P. Zheng, S. F. Armanini, and M. Kovac, ≪Unmanned aerial sensor placement for cluttered environments≫, IEEE Robotics and Automation Letters **5**, 6623–6630 (2020).

[3]P. Y. Li and R. Horowitz, ≪Passive velocity field control (pvfc). part i. geometry and robustness≫, IEEE Transactions on Automatic Control **46**, 1346–1359 (2001).

[4]P. Y. Li and R. Horowitz, ≪Passive velocity field control of mechanical manipulators≫, IEEE Transactions on robotics and automation **15**, 751–763 (1999).

[5]C. McInnes, ≪Velocity field path-planning for single and multiple unmanned aerial vehicles≫, The Aeronautical Journal **107**, 419–426 (2003).

[6]F. Ruggiero, V. Lippiello, and A. Ollero, ≪Aerial manipulation: A literature review≫, IEEE Robotics and Automation Letters **3**, 1957–1964 (2018).