Vision-based motion control for the ACROBAT free-flyer prototype

VIEGAS, Guilherme
guilherme.viegas@tecnico.ulisboa.pt
90090

February 21, 2022

***Abstract*—The use of autonomous robots with the capability of manufacturing and assembling of large structures in micro-gravity is a growing need. Robots with such capabilities must be able to safely and reliably move along a given trajectory. This project develops a software system capable of performing pose stabilization of a 3 degree of freedom free-flying robot (ACROBAT), using visual markers. The visual markers are used to estimate the position and attitude of the robot. A Proportional-Derivative controller is designed for the translational portion of the robot, and an exponentially convergent $SO(3)$ controller designed for the rotational portion of the robot. The dynamic model of the ACROBAT robot is used to compute the actuation signals needed for moving and rotating the robot, sent directly to real robot actuators as Pulse-Width Modulation (PWM) signals.**

***Keywords*—** Space robotics, Free-Flyer, Pose Stabilization, Visual Markers

II. Introduction

The next few years will be of much importance for the space exploration subject. With a growing drive towards reaching Mars and exploring the deeper space, in-orbit robotized manufacturing and assembly of large structures in space is becoming a growing need. The deployment of autonomous robots, more specifically, aerial free-flyers in micro-gravity environments could is a very interesting solution to these kind of problems. These forms of robots by being capable of holonomic movement and being packed with interesting sensor capabilities and some even with manipulators like 2 degrees of freedom (DoF) manipulators pose great potential for automating simple tasks as logistics and for repair, manufacturing and assembly of certain structures.

One recently developed free-flyer is the ACROBAT [Vale et al. 2021], seen in Figure 1, that uses propellers as the propulsion system, which limit its movement to pressurized micro-gravity environments like the International Space Station (ISS) and an attached six DoF six Revolute-Spherical-Spherical (RSS) parallel manipulator, similar to the one proposed in [Pierrot, Dauchez, and Fournier 1991], to the robot body. For ground testing, a reduced version was built containing only 3 propellers: the attitude is constrained to yaw movement and the translational motion is constrained to a horizontal plane (using an air-bearing table), thus the kinematic space has 3 DoF.

The goal of this work is to perform vision-based motion control using visual markers such as AR tags, like the ones seen in Figure 2 .This markers are used to estimate the position and attitude of the robot and the final goal is to perform pose stabilization and trajectory tracking. This is achieved via a closed-loop feedback Proportional-Derivative (PD) controller for the translational part of the movement and a an exponentially convergent $SO(3)$ controller proposed in [Lee 2012] for the attitude movement, that uses the position and attitude estimates from the AR tags. Only ground tests are done, so only yaw rotation and translation around an horizontal plane are permitted, and no parallel manipulator will be attached to the ACROBAT. The report is structured as follows: III presents the concept of free-flying robots naming some interesting cases. IV and V develops the specific case of the ACROBAT, which will be the robot used for this project, presenting its propulsion model and its dynamic model respectively, making the needed division between the supposed 6 DoF ACROBAT and the existing testbed with only 3 DoF. VI proposes the feedback controller to be applied to the translational and rotational modes. VII describes briefly the chosen visual markers and their capabilities. VIII presents the software structures and some approximations that were done in order to test the project in the real ACROBAT. IX shows the obtained results, testing the project both with and without the physical ACROBAT. X wraps up this report by drawing some conclusions on the work done and future ways of extending this work.

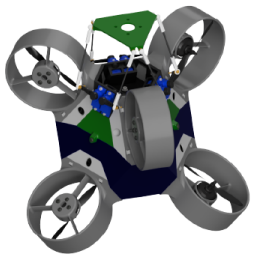


Fig. 1: ACROBAT cad model

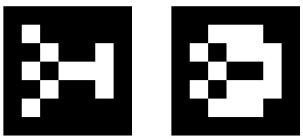


Fig. 2: Two example AR tags produced by the Aruco marker package

III. Free-Flyers

Assistive Free-flyer robots (AFF's), defined in [Albee and Hernandez 2021] as

> *"Mobile satellite platforms that operate using both thrusters and internal reconfiguration like a manipulator arm",*

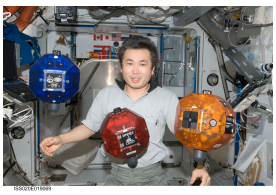
have had its dynamics analysed by Dubowsky and others in the early 1990s [Dubowsky and Papadopoulos 1993]. Due to the subtle (but harsh) conditions of working in space, many real-life testbeds were developed in order to controllably study and tackle this problems.

The Synchronized Position Hold Engage Reorient Experimental Satellites, or SPHERES [Mohan et al. 2009], as seen in Figure 3a, was a testing facility, deployed in the ISS from 2006 to 2019, mainly for flight formation experiments, consisting of 3 identical round-shaped satellites that used CO2 compressed gas as a propulsion mechanism. Developed to mature flight formation, docking, path planning and other experimental algorithms. More than 100 experiments were done within the SPHERES ecosystem which also included a glass table in an MIT lab for verification experiments.

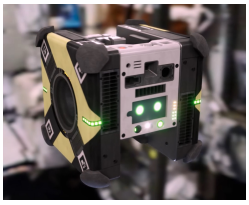
With the need of a more powerful and better sensor equipped free-flyer, NASA developed the Astrobee [Bualat et al. 2018], as seen in Figure 3b, being the currently deployed testing platform of this kind in the ISS. The Astrobee system is composed of three Astrobees, a docking station capable of charging them and a ground data system. Being an upgrade of the SPHERES project and drawing inspiration from the AERCam [Williams and Tanygin 1998] and PSA [Dorais and Gawdiak 2003] projects, it is capable of docking and undocking autonomously, flying farther and faster, has better sensors like image and depth cameras, more computational power leveraged by three separate (and singly-purposed) micro-processors and can make use of a 2 DoF manipulator either for experiments or to perch to ISS's handrails. The Astrobee Robot Software, built on top of ROS, exists as an open-source project [1] and brings the add-on of a very realistic Astrobee simulator built on ROS and Gazebo.

The Space CoBot is another free-flyer robot, capable of IVA experiments due to its propeller based propulsion [Roque and Ventura 2016]. It has a fully omnidirectional (holonomic) motion meaning 6 degrees of freedom and a modular design that enables the extension with other research components or even the attachment to another Space CoBot as seen in Figure 3c. Developed with the goal of broadening the range of services that such types of free-flyers provide, its vision is not only to improve collaboration between onboard and Earth crews but also in small object management like tracking and manipulating objects such as pens, screws, etc. Currently the robot is developed and exists for testing either in a granite table and/or in a 4-ring gimbal.

The ACROBAT proposed in [Vale et al. 2021], in Figure 1, is a designed free-flyer with six DoF movement and an attached six DoF six Revolute-Spherical-Spherical parallel manipulator intended for cooperative in-orbit manufacturing and assembly tasks. Although not fully built, a three DoF version of it was built, being constrained to a planar movement and yaw rotation.



(a) 3 SPHERES in ISS



(b) Astrobee in ISS

(c) Two Space CoBots attached

Fig. 3: Existing Free-Flyer test-beds

[1] https://github.com/nasa/astrobee

IV. ACROBAT'S PROPULSION MODEL

Before diving into the vision-based motion control problem, it is crucial to describe the ACROBAT dynamic model, being only relevant the propulsion model and not the manipulator model as no manipulator will be attached to the ACROBAT when testing.

As proposed by [Vale et al. 2021], the ACROBAT is constituted by six propellers, displaced such that the kinematics are holonomic, meaning the robot can freely move with six DoF, and so this work will follow the same dynamic model as described there, doing the needed changes for the specific 3 DoF version. A single propeller i, if attached to the body frame $\mathcal{B}$ that is coincident with the Center of Mass (CoM), generates a thrust $\mathbf{F_i}$ and torque $\mathbf{M_i}$ on the ACROBAT while rotating at speed n_i (in revolutions per second). $\mathbf{F_i}$ only depends on the propeller properties and thrust

$$\mathbf{F_i} = f_i \mathbf{u_i} \qquad f_i = K_1 |n_i| n_i \tag{1}$$

with $\mathbf{u_i}$ being a unit vector aligned with the propeller's axis of rotation, as seen in Figure 4, and K_1 is a propeller constant given by

$$K_1 = \rho D^4 C_T \tag{2}$$

where ρ is the air density, D is the propeller diameter and C_T is the thrust coefficient.

$\mathbf{M_i}$ is caused by the propeller non-central thrust $\mathbf{F_i}$ and rotation torque τ_i

$$\mathbf{M_i} = \mathbf{r_i} \times \mathbf{F_i} - \tau_i \mathbf{u_i} \qquad \tau_i = w_i K_2 |n_i| n_i \tag{3}$$

with $\mathbf{r_i}$ being the propeller position relative to $\mathcal{B}$, w_i being used to define a positive or forward thrust, where -1 is used if the propeller rotates clockwise and 1 otherwise and K_2 another propeller constant given by

$$K_2 = \frac{\rho D^5}{2\pi} C_P \tag{4}$$

where the remaining not described parameter is C_P which is the power coefficient. K_1 and K_2 are here considered equal for both directions of rotation of the propeller.

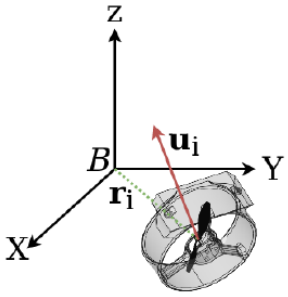


Fig. 4: Notation of a single propeller, relative to the body's CoM

Considering an actuation signal vector $\mathbf{q}$, with the i-th actuation signal defined as $q_i = |n_i| n_i$, one can define the force and torque as

$$\begin{bmatrix} \mathbf{F_i} \\ \mathbf{M_i} \end{bmatrix} = \mathbf{a_i} q_i, \tag{5}$$

with

$$\mathbf{a_i} = \begin{bmatrix} K_1 \mathbf{u_i} \\ K_1 \mathbf{r_i} \times \mathbf{u_i} - w_i K_2 \mathbf{u_i} \end{bmatrix} \tag{6}$$

One can divide the terms by the constant K_1, resulting in

$$\mathbf{a_i'} = \begin{bmatrix} \mathbf{u_i} \\ \mathbf{r_i} \times \mathbf{u_i} - w_i \frac{K_2}{K_1} \mathbf{u_i} \end{bmatrix}, \tag{7}$$

so that it is now possible to simplify this expression, because of the size of the propellers (4") which lead to $\frac{K_2}{K_1}$ taking values of the magnitude of 10^{-2} by making the approximation of $\frac{K_2}{K_1} \approx 0$, leading to

$$\mathbf{a_i'} = \begin{bmatrix} \mathbf{u_i} \\ \mathbf{r_i} \times \mathbf{u_i} \end{bmatrix}, \tag{8}$$

Considering that the ACROBAT possesses six propellers, the total force ($\mathbf{F} \in \mathbb{R}^3$) and torque ($\mathbf{M} \in \mathbb{R}^3$) can be given in matrix form as

$$\begin{bmatrix} \mathbf{F} \\ \mathbf{M} \end{bmatrix} = \begin{bmatrix} \mathbf{a_1} \dots \mathbf{a_6} \end{bmatrix} \begin{bmatrix} q_1 \\ \vdots \\ q_6 \end{bmatrix} = \mathbf{Aq} \tag{9}$$

$\mathbf{A}$, the actuation matrix, is a square matrix of size $\mathbb{R}^{6\times6}$ and full rank meaning it is invertible. This property enables to establish the following relationship

$$\mathbf{q} = \mathbf{A^{-1}} \begin{bmatrix} \mathbf{F} \\ \mathbf{M} \end{bmatrix} \tag{10}$$

The actuation signal, $\mathbf{q}$ has to be converted from rotations per second to the pulse-width modulation (PWM) domain. Using the python *pigpio*[2] library to send the PWM signals, it follows a relation that is used conventionally for servo motors, although usually applied to other examples as the case of brushless motors. This convention states a usual servo motor expects to be updated every 20 ms with a pulse between 1 ms and 2 ms, or in other words, between a 5 and 10 duty cycle on a 50 Hz waveform. This means that a 1.5ms pulse centers the servo on the 90º or for the case of the brushless motor, sets it in a centered position (not rotating). A 2.5ms pulse makes the motor rotate at its full anti-clockwise speed and a 0.5 ms pulse at its full clockwise speed. Sending a 0 valued pulse turns off the motor, meaning to restart any motor movement one has to first send a 1.5ms pulse to arm the ESC and then one can start sending any other pulse to actuate the motors. With this said, one can provide the relation between the rps signal and the real pwm actuation signal, regarding that both F^{max} = 2 and M^{max} = 2 as

$$\begin{cases} q_{pwm} = 1500 - \frac{2 \cdot q_{rps}}{1000} & \text{if } q_{rps} \leq 0 \\ q_{pwm} = 1500 + \frac{2 \cdot q_{rps}}{1000} & \text{if } q_{rps} > 0 \end{cases} \tag{11}$$

The $\mathbf{r_i}$ and $\mathbf{u_i}$ parameters were designed and obtained via a multi-objective optimization problem, which will not be addressed here and can be seen in detail in [Vale et al. 2021]. With those parameters and with $\mathbf{r_i}$ and $\mathbf{u_i}$, from inspection of Figure 5, can written as

$$\mathbf{r_i} = \begin{bmatrix} cos(\gamma_i)sin(\theta_i) \\ sin(\gamma_i)sin(\theta_i) \\ cos(\theta_i) \end{bmatrix} \qquad \mathbf{u_i} = \begin{bmatrix} cos(\alpha_i)sin(\beta_i) \\ sin(\alpha_i)sin(\beta_i) \\ cos(\beta_i) \end{bmatrix} \tag{12}$$

with $\theta_i \in [0, \pi], \gamma_i \in [0, 2\pi], \beta_i \in [0, \pi], \alpha_i \in [0, 2\pi]$ presenting only the final obtained values

$$\mathbf{A} = \begin{bmatrix} 2.18 & -9.99 & -1.12 & 1.61 & 99.8 & -0.0649 \\ 99.9 & 1.86 & 99.9 & 1.91 & 1.90 & 5.80 \\ -2.22 & 1.28 & 1.08 & 99.9 & -5.62 & 99.8 \\ -99.9 & 1.34 & 99.8 & -3.32 & 5.83 & -1.71 \\ 2.18 & 3.22 & 1.19 & -99.9 & -11.3 & 99.6 \\ 0.0 & 99.9 & -5.99 & 1.97 & 99.8 & -5.78 \end{bmatrix} \times 10^{-2} \tag{13}$$

Because the tested robot only has three out of six propellers available and considering a displacement of the vectors r_i and u_i as described in Figure 6, the final $\mathbf{A}$ matrix, for the 3 DoF case is

$$\mathbf{A} = \begin{bmatrix} 0.2588 & -0.9659 & 0.7071 \\ -0.9659 & 0.2588 & 0.7071 \\ -0.8529 & -0.8529 & -0.8529 \end{bmatrix} \tag{14}$$

[2]https://abyz.me.uk/rpi/pigpio/

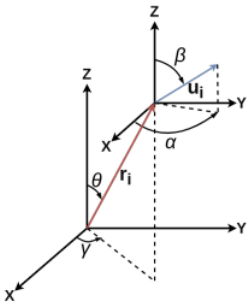


Fig. 5: $\mathbf{r_i}$ and $\mathbf{u_i}$ relative to the body's CoM

and the same relation in (10) exists because in this 3 DoF domain the force $\mathbf{F}$ only has the F_x and F_y components and the torque $\mathbf{M}$ only has the M_z component.

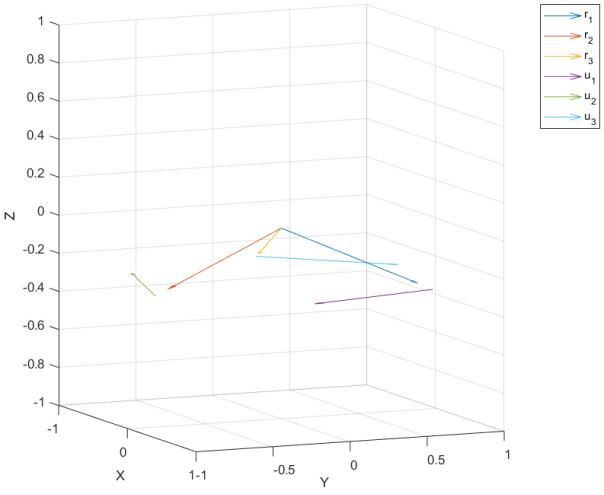


Fig. 6: Vectors r_i and u_i displacement

V. ACROBAT'S DYNAMIC MODEL

Applying the Newton-Euler equations of motion to the case study of an holonomic free-flyer, one can define its state-space and state dynamics for the six DoF use-case first, as in [Ventura, Roque, and Ekal 2018], for generality purposes, and then diving into the three DoF use-case. Denoting the position and velocity of the body frame $\mathcal{B}$, centered on the vehicle CoM and aligned with respect to the inertial frame $\mathcal{I}$ as x and v, the rotation matrix of frame $\mathcal{B}$ with respect to $\mathcal{I}$ as $\mathbf{R}$, and the angular velocity of the vehicle in the body frame $\mathcal{B}$ as ω, one as the dynamic system described as

$$\begin{cases} \dot{x} = v \\ m\ddot{v} = \mathbf{R}F \\ \dot{\mathbf{R}} = \mathbf{R}\mathbf{S}(\omega) \\ \mathbf{J}\dot{\omega} = M - \omega \times \mathbf{J}\omega \end{cases} \tag{15}$$

where the constants m and $\mathbf{J}$ are the robot's mass and moment of inertial matrix. $\mathbf{S}(\omega)$ is the skew-symmetric matrix defined by

$$\mathbf{S}(\omega) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{16}$$

for $\omega = \begin{bmatrix}\omega_x & \omega_y & \omega_z\end{bmatrix}$ Being this a transformation from $\mathbb{R}^3$ to an SO(3) skew-symmetric matrix, one can also define the inverse transformation, S^{-1} that maps an SO(3) skew-symmetric matrix to the corresponding vector in $\mathbb{R}^3$.

The 3 DoF dynamic model of the case study of a free-flyer on a granite table, having its movement constrained to an horizontal plane and the yaw rotational movement is a simplification of the aforementioned 6 DoF dynamic model. The system state is then composed of 6 state variables instead of 12, as in

$$\begin{cases}\mathbf{r} = \begin{bmatrix}r_x & r_y\end{bmatrix}^T \\ \mathbf{q} = \begin{bmatrix}q_z\end{bmatrix}^T \\ \mathbf{v} = \begin{bmatrix}v_x & v_y\end{bmatrix}^T \\ \omega = \begin{bmatrix}\omega_z\end{bmatrix}^T\end{cases} \qquad \mathbf{x} = \begin{bmatrix}\mathbf{r} & \mathbf{q} & \mathbf{v} & \omega\end{bmatrix} \tag{17}$$

VI. Motion Controller

Due to its holonomic design, the translational and rotational modes can be decoupled such that

$$\mathbf{q} = \mathbf{b^T F} + \mathbf{c^T M} \tag{18}$$

where $\mathbf{b}$ and $\mathbf{c}$ come from the fact that matrix $\mathbf{A^{-1}}$ can be rewritten as the composition of those matrices

$$\mathbf{A^{-1}} = \begin{bmatrix}\mathbf{b}_1^T & \mathbf{c}_1^T \\ \vdots & \vdots \\ \mathbf{b}_6^T & \mathbf{c}_6^T\end{bmatrix} \qquad \text{with} \mathbf{b_i}, \mathbf{c_i} \in \mathbb{R}^3, \tag{19}$$

This opens the possibility to develop two independent controllers, one for the translational mode and another for the rotational mode. Following the used controllers in [Roque and Ventura 2016], a Proportional-Derivative (PD) controller is used for the translational mode, because its linearized system

$$\begin{cases}\dot{x} = v \\ \dot{v} = p \\ F = m\mathbf{R}^T p\end{cases} \tag{20}$$

is diagonal and of second order, meaning a PD controller is enough to ensure exponential convergence. So, one just needs to design a feedback PD controller for p:

$$\begin{cases}e_x = x - x_d \\ e_v = v - v_d \\ p = -k_x e_x - k_v e_v\end{cases} \tag{21}$$

where x_d and v_d are the desired position and velocity vectors with respect to the inertial frame $\mathcal{I}$, and k_x and k_v are the proportional and derivative gains of the PD controller, tuned later on when testing.

For the attitude control it is followed the exponentially convergent SO(3) controller proposed in [Lee 2012]:

$$\begin{cases}e_r = \frac{1}{2\sqrt{1+tr[\mathbf{R}_d^T\mathbf{R}]}} S^{-1}(\mathbf{R}_d^T\mathbf{R} - \mathbf{R}^T\mathbf{R}_d) \\ e_\omega = \omega - \mathbf{R}^T\mathbf{R}_d\omega_d \\ M = -k_R e_R - k_\omega e_\omega + S(\mathbf{R}^T\mathbf{R}_d\omega_d)\mathbf{J}\mathbf{R}^T\mathbf{R}_d\omega_d + \mathbf{J}\mathbf{R}^T\mathbf{R}_d\dot{\omega}_d\end{cases} \tag{22}$$

where $\mathbf{R}_d$ is the rotation matrix of the desired attitude, ω_d is the desired angular velocity vector and with k_R and k_ω, also tuned later on when testing.

VII. Aruco Markers

Although many ways to tackle the problem of position and attitude tracking exist nowadays, one of the cheapest and easier ways to implement it is by using Augmented Reality tags (AR tags). This kind of tags, with some examples seen in Figure 2, are black and white squared images with some patterns within a black border that are purposely created for this types of problems to allow a decrease in processing time and a good position and attitude estimation. Usually represented by matrices of 5x5 size and generated differently by different algorithms, they are especially suitable for pose estimation because of its predefined and known sizes (suitable for the position estimation) and its capability of defining a reference frame for the AR tag (suitable for the attitude estimation), as seen in Figure 7. The used algorithm is defined in the *ArUco* package[3], integrated in the *opencv* python library and with a package also developed for ROS. This package was chosen, out of many available, due to its thorough documentation and easy implementation.

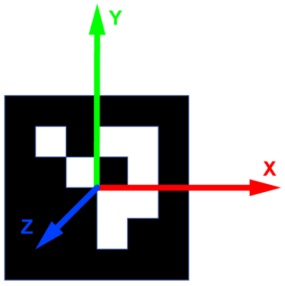


Fig. 7: Example of AR tag with the defined reference frame

The *ArUco* package comes with different dictionaries, which are sets of tags, built for different functionalities. The main differences between dictionaries are with respect to the size of its tags, where a dictionary containing small tags is useful for close-range purposes, where the robot is very close to the tag, and vice-verse and the other difference as to do with the number of tags per dictionary, because although some projects may need to use many tags at the same time, this can also lead to more errors in identifying tags. For the case of this project, where only one tag will be used and where the robot might come close to it, the "DICT_5X5_50" was chosen, having the smallest number of tags inside it and each one being a 5cm by 5cm shaped square.

VIII. Implementation Details

Before diving into the details of the software implementation, one should describe briefly the hardware needed to bring this project to real life:

- ACROBAT structure, as in Figure 1 where all the remaining hardware is mounted, noting that for this project, the 6 DoF manipulator is not used and also the used version of the ACROBAT only has 3 of the 6 propellers;
- Propellers and respective Electronic Speed Controllers (ESC's) which adds the actuation capabilities to the ACROBAT and enables to easily control the motor's actuation based only on simple PWM pulses;
- An *USB Logitech Webcam* attached to the ACROBAT to enable video capturing and therefore enabling the localization of the AR tags;
- A *RaspberryPi 3 - Model B* that will be used as the robot's board computer
- A LiPo battery capable of powering both 3 motors and the RaspberryPi

The complete setup can be seen in Figure 8.

The developed software[4], fully implemented in Python, follows

[3] https://docs.opencv.org/3.4/d5/dae/tutorial_aruco_detection.html

[4] https://github.com/Guilherme-Viegas/PositionAndAttitudeEstimation3DoF_Free_Flyer

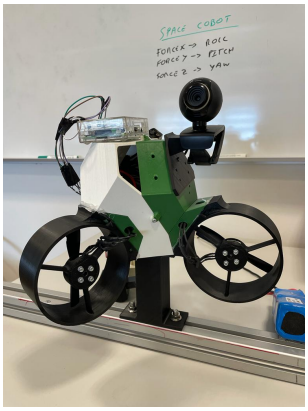


Fig. 8: Complete setup of ACROBAT to be tested

a very straightforward implementation, having in the Appendix a further explanation on how to use it. First one should take a moderate amount of photographs (ideally more than 10) of what is called a *calibration chessboard*, which can be seen in Figure 9. This is used to compute the camera calibration parameters, specifically the camera intrinsic parameters and distortion coefficients. This parameters remain fixed unless the camera optic is modified, thus the camera calibration only needs to be performed once. Storing the taken photographs into a folder named */workdir/* and running the python script named *generate_cam_calib_file.py* generates a .npz file with all the computed camera parameters, which will later be used to better localize the AR tags.

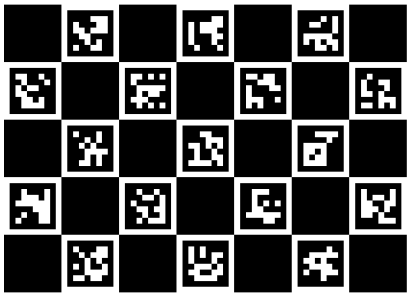


Fig. 9: Calibration Chessboard

Having the camera correctly calibrated, now the main code shall be explained, which is divided into three separate files

- *main.py* where the main function is defined, loading the camera parameters, reading the input frames from the mounted camera and calling the needed functions to 1) Perform the localization of the AR tag and estimate both position and attitude of the ACROBAT relative to the AR tag; 2) Call the controller functions to generate values for force and torque based on the ACROBAT's current position and attitude and 3) Convert the force and torque values to pwm signals that can be directly sent to the ESC's to actuate on the motors;
- *aruco_tracker.py* defining the possible Aruco library dictionaries so that the software is capable of knowing what AR tag one expects to read, and also defining some utility functions specific to the Aruco library such as checking if the chosen dictionary is valid;
- *motor_control.py* which by connecting to the RaspberryPi GPIO pins and making a mapping between the intended force and torque inputs and the corresponding pwm signals that are needed, sends those signals directly to the motor ESC's;
- *controller.py* responsible for computing the needed force and torque signals based on the current position and attitude errors as explained in Chapter VI

A. Controller Specifics

Some important remarks have to be done regarding the implemented controller.

First, using the *ArUco* library to compute the pose and attitude of the robot one has to be careful with the used frames. First, noted in the libraries documentation, the estimates of the position and attitude are of the tag relative to the camera, not otherwise as one would want. Regarding the position, although not so problematic because wanting the robot to be, for example 20cm in front of the marker, is the same as wanting the marker to be 20cm in front of the robot, it is important to observe that wanting the robot in front of the marker by 20cm means defining a desired position of $\mathbf{r} = \begin{bmatrix}0 & 0 & 0.2\end{bmatrix}^T$, because its the Z axis in the marker frame that is orthogonal to it. When it comes to the attitude, although not so simple, it is just a matter of making the correct transformation between frames, with the careful note that the camera frame is defined as in Figure 10.

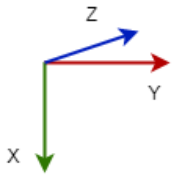


Fig. 10: Camera Frame

Comparing the frames between Figure 7 and Figure 10, the final rotation matrix that serves as input for the controller computations can be found with

$$R = R_x(\pi)R_{marker} = \begin{bmatrix}1 & 0 & 0\\ 0 & cos(\pi) & -sin(\pi)\\ 0 & sin(\pi) & cos(\pi)\end{bmatrix} R_{marker} \quad (23)$$

with $R_x(\pi)$ being a rotation matrix of π rads around the X axis and R_{marker} the generated rotation matrix from the *ArUco* library between the marker and the camera.

Secondly, what the *ArUco* library outputs as attitude estimation is a rotation vector $\in \mathcal{R}^3$ instead of the usual (and needed) rotation matrix $\in SO(3)$. To make this conversion, the *Rodrigues* formula is used

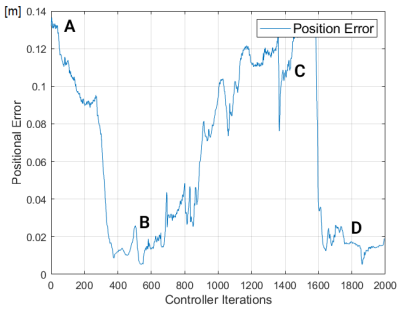
$$\mathbf{R}_{marker} = cos(\theta)I + (1 - cos(\theta))\mathbf{r}\mathbf{r}^T + sin(\theta)\begin{bmatrix}0 & -r_z & r_y\\ r_z & 0 & -r_x\\ -r_y & r_x & 0\end{bmatrix} \quad (24)$$

IX. Experimental Results

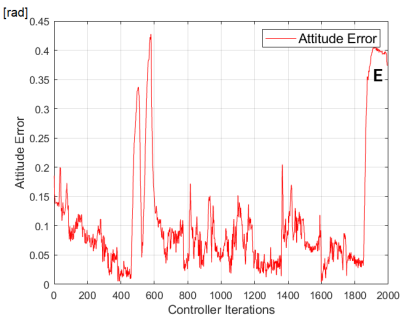
As the available robot has only 3 DoF, being constrained to movement in the horizontal plane and the yaw rotation, some simplifications are made to the 6 DoF free-flyer dynamics previously mentioned, namely by reducing the actuation matrix to only correspond to the only available 3 propellers as in (14) and change some of the controller computations.

With the initial goal of performing pose stabilization, one sets the desired state vector to be exactly 20cm in front of the AR tag. With this in mind, first an experiment using only the camera and moving the visual marker around, the computed values for the positional and rotational errors were taken, taking also the computed actuation signals for the three propellers. The positional error,as function of the loop iteration,) is seen in Figure 11a and the attitude error is seen in Figure 11b. Marked with the letters from 'A' to 'E' are important

points of the visual marker movement, that correspond respectively to the Figure 12 representations. 'A' corresponds to a starting point of the visual marker at the left of the camera, Figure 12a, taking a relatively big positional error, while maintaining a small attitude error. 'B', in Figure 12b happens when the visual marker is going from left to right, here taking some big leaps in the attitude error, mainly due to oscillations while moving the target. 'C', is when the visual marker is at the right of the camera, taking again bigger positional errors. 'D' happens when the visual marker is almost perfectly positioned and aligned with the camera, shown in Figure 12d. Finally, 'E' shows a bigger attitude error due to an intentional yaw movement, as seen in Figure 12e.



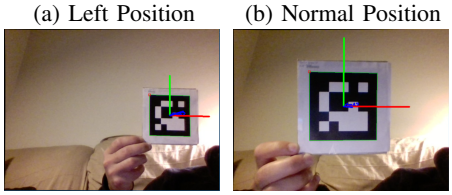
(a) Position error



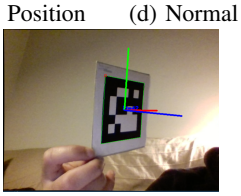
(b) Rotation error

Fig. 11: Position and Attitude errors along a performed visual marker movement

(a) Left Position (b) Normal Position



(c) Right Position (d) Normal Position



(e) Yaw Rotation

Fig. 12: Movement of the visual marker

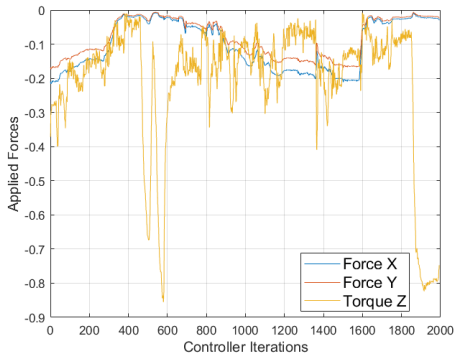


Fig. 13: Corresponding Forces and Torque to the visual marker motion

In Figure 13 one can see the computed Forces, in the X and Y directions and the Torque, around the Z axis, corresponding to the visual marker motion. In Figure 14 one can also see the computed actuation signals for each motor corresponding to the visual marker motion, proving the physical robot tries to compensate for the error between its position and attitude (of the camera actually) and the visual marker position and attitude. Here, because the marker was never closer to the camera than the desired distance and the yaw movement of the visual marker was always in the same direction, the actuation signals never generated a reversed propeller movement (signal pulse less than 1.5ms), although capable of that.

Tests done on the physical robot were very limited, because one of the motors had a strange malfunction working only on a very low rotations per second value, while the other two worked fine. Although this limitation, it was possible to check the variation of the motors speed when a change on the visual marker position and/or attitude was detected.

X. Conclusion

This project addressed the problem of pose stabilization of a 3 degrees of freedom free-flyer using visual markers to estimate its position and attitude. For this, a software code written in python detected the visual marker and estimated the position and attitude of the robot relative to the marker, generated a feedback control law using two different controllers; one Proportional-Derivative controller for the translational mode and an exponentially convergent $SO(3)$ controller for the rotational mode. From this, the corresponding actuation signals for each motor were computed based on the specific propulsion model and dynamics of the robot system. Although this work was specifically designed for the 3 DoF version of the ACROBAT free-flyer, it could be easily extended for the 6 DoF version, needing only to change the defined actuation matrix and some minor changes corresponding to a different working dimension.

Future work is seen as great motivation, because this work could be interestingly extended to trajectory tracking of the robot and maybe working with more than one visual marker. The use of an IMU system could be of great use, as for this case there was no possibility to use one, and using more realistic measurements for certain parameters such as the robot's mass or moments of inertia could be used to better improve the controller's performance. To improve performance, using a more powerfull board computer could be used such as the *Intel UpBoard* to speed up the controller loop. Tests on the test table to validate the controller are expected to be done in short time, which will be a major landmark for the ACROBAT project.

Contents

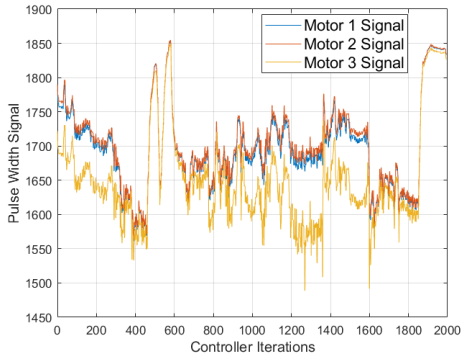


Fig. 14: Corresponding motor actuation signals to the visual marker motion

List of Figures

References

Albee, Keenan and Alejandro Cabrales Hernandez (Jan. 2021). "The Case for Parameter-Aware Control of Assistive Free-Flyers". In: *AIAA 2018-2021 - Session:Motion Planning, Sensing, and Operations for Aerospace Robotic Systems II*. DOI: https://doi.org/10.2514/6.2021-2018.

Bualat, Maria G. et al. (May 2018). "Astrobee: A New Tool for ISS Operations". In: *Proceedings of the 15th International Conference on Space Operations*. DOI: doi=10.2514/6.2018-2517.

Dorais, Gregory A. and Yuri Gawdiak (Mar. 2003). "The personal satellite assistant: an internal spacecraft autonomous mobile monitor". In: *Proceedings of the 2003 IEEE Aerospace Conference*. DOI: 10.1109/AERO.2003.1235064.

Dubowsky, S. and E. Papadopoulos (1993). "The kinematics, dynamics, and control of free-flying and free-floating space robotic systems". In: *IEEE Transactions on Robotics and Automation* 9.5, pp. 531–543. DOI: 10.1109/70.258046.

Lee, Taeyoung (2012). "Exponential stability of an attitude tracking control system on SO(3) for large-angle rotational maneuvers". In: *Systems & Control Letters* 61.1, pp. 231–237. ISSN: 0167-6911. DOI: https://doi.org/10.1016/j.sysconle.2011.10.017. URL: https://www.sciencedirect.com/science/article/pii/S0167691111002829.

Mohan, Swati et al. (Oct. 2009). "SPHERES Flight Operations Testing and Execution". In: *Acta Astronautica 65* 7-8, pp. 1121–1132.

Pierrot, F., P. Dauchez, and A. Fournier (1991). "HEXA: a fast six-DOF fully-parallel robot". In: *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*, 1158–1163 vol.2. DOI: 10.1109/ICAR.1991.240399.

Roque, Pedro and Rodrigo Ventura (2016). "A Space CoBot for personal assistance in space stations". In: *IJCAI-2016 Workshop on Autonomous Mobile Service Robots, New York City, NY, USA*.

Vale, João et al. (July 2021). "A Multi-Objective Optimization Approach to the Design of a Free-Flyer Space Robot for in-orbit Manufacturing and Assembly". In: *AeroBest 2021 - International Conference on Multidisciplinary Design Optimization of Aerospace Systems*, pp. 517–536.

Ventura, Rodrigo, Pedro Roque, and Monica Ekal (Oct. 2018). "Towards an Autonomous Free-flying Robot Fleet for Intra-vehicular Transportation of Loads in Unmanned Space Stations". In: *Proceedings of the 69th International Astronautical Congress (IAC), Bremen, Germany*.

Williams, Trevor and Sergei Tanygin (1998). "On-orbit engineering tests of the AERCam Sprint robotic camera vehicle". In: *Spaceflight Mechanics 1998*, pp. 1001–1020.

Appendix

Some simple steps are needed in order to use the code.

1) Download and save the code in the board computer, for example the Raspberry Pi
2) Install the need packages, namely the *opencv*, *numPy* and *pigpio* libraries
3) Connect the ESC pins to the 12, 13 and 18 GPIO pins of the Raspberry Pi, connecting also the corresponding grounds, being these pins described in Figure 15
4) Connect the camera and confirm that the defined video input in the main file is the correct camera. Usually "/dev/video0" or "/dev/video1 will work. This can be checked by running "cd /dev/", writing "video" and tapping the *Tab* button will show the existing connected video capture devices
5) Connect the battery to the Power Distribution Board

6) Start the pigpio daemon by running the command ”sudo pigpiod”
7) Starting the software by running ”python ./main.py” (this project was done for python2, so there are no guarantees that with python3 will work)

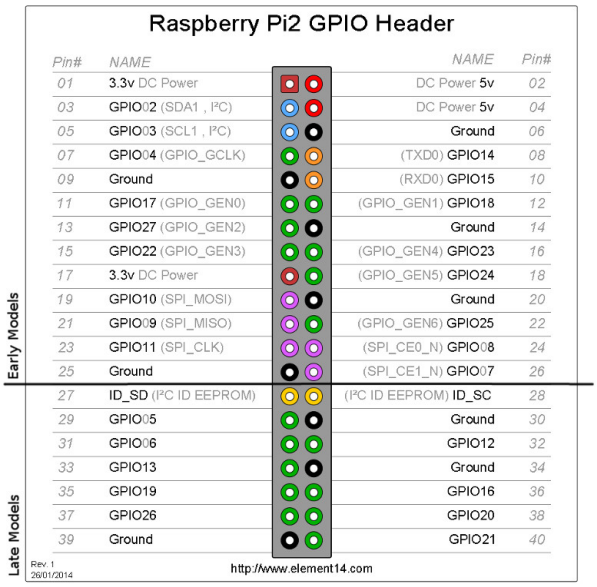


Fig. 15: Raspberry Pi GPIO pins