Physically based simulation in Computer Graphics

Milestone presentation Group 11

Alessandro Petitti

Luca Monegaglia

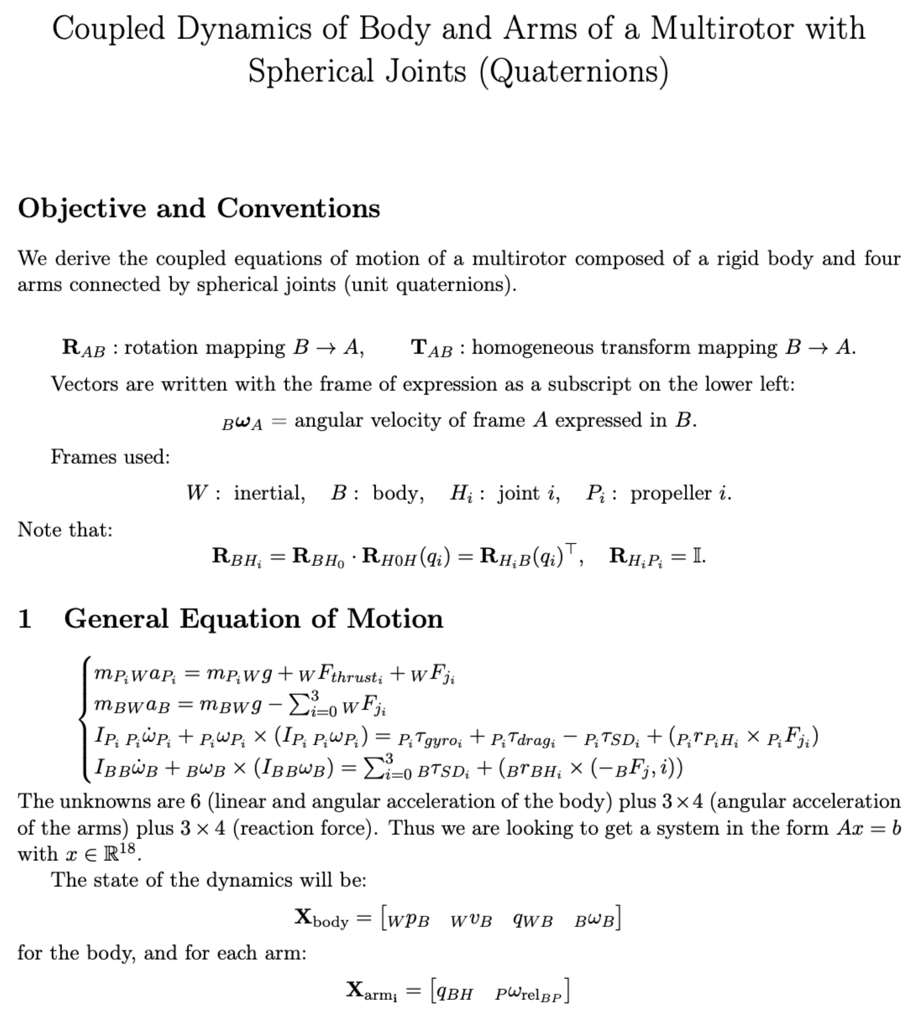
Gennaro Guidone

Outlook:

- Current state
  - Model
  - Integrators
  - Visualization
- Encountered Problems
  - Siff dynamics and hard modeling
  - Slow integration problem
- Next Steps
  - Handling contacts

Current state of the project - model

Modeling has been done, equation of motion of the multibody systems have been derived, staring from Newton and Euler equations.
The acceleration of the multiples body resulted to be coupled, due to the visco-elastic joint connecting them, resulting into a 18x18 matrix that needs to be inverted at each timestep.



Coupled Dynamics of Body and Arms of a Multirotor with Spherical Joints (Quaternions)

Objective and Conventions

We derive the coupled equations of motion of a multirotor composed of a rigid body and four arms connected by spherical joints (unit quaternions).

$$\mathbf{R}_{AB} : \text{rotation mapping } B \to A, \qquad \mathbf{T}_{AB} : \text{homogeneous transform mapping } B \to A.$$

Vectors are written with the frame of expression as a subscript on the lower left:

$$_B\boldsymbol{\omega}_A = \text{angular velocity of frame } A \text{ expressed in } B.$$

Frames used:

$$W: \text{ inertial}, \quad B: \text{ body}, \quad H_i: \text{ joint } i, \quad P_i: \text{ propeller } i.$$

Note that:

$$\mathbf{R}_{BH_i} = \mathbf{R}_{BH_0} \cdot \mathbf{R}_{H0H}(q_i) = \mathbf{R}_{H_iB}(q_i)^\top, \quad \mathbf{R}_{H_iP_i} = \mathbb{I}.$$

1 General Equation of Motion

$$\begin{cases} m_{P_i W} a_{P_i} = m_{P_i W} g + {}_W F_{thrust_i} + {}_W F_{j_i} \\ m_{BW} a_B = m_{BW} g - \sum_{i=0}^{3} {}_W F_{j_i} \\ I_{P_i} \, {}_{P_i}\dot{\omega}_{P_i} + {}_{P_i}\omega_{P_i} \times (I_{P_i} \, {}_{P_i}\omega_{P_i}) = {}_{P_i}\tau_{gyro_i} + {}_{P_i}\tau_{drag_i} - {}_{P_i}\tau_{SD_i} + ({}_{P_i}r_{P_iH_i} \times {}_{P_i}F_{j_i}) \\ I_{BB}\dot{\omega}_B + {}_B\omega_B \times (I_{BB}\omega_B) = \sum_{i=0}^{3} {}_B\tau_{SD_i} + ({}_B r_{BH_i} \times (-{}_B F_j, i)) \end{cases}$$

The unknowns are 6 (linear and angular acceleration of the body) plus 3×4 (angular acceleration of the arms) plus 3×4 (reaction force). Thus we are looking to get a system in the form $Ax = b$ with $x \in \mathbb{R}^{18}$.

The state of the dynamics will be:

$$\mathbf{X}_{\text{body}} = \begin{bmatrix} {}_W p_B & {}_W v_B & q_{WB} & {}_B\omega_B \end{bmatrix}$$

for the body, and for each arm:

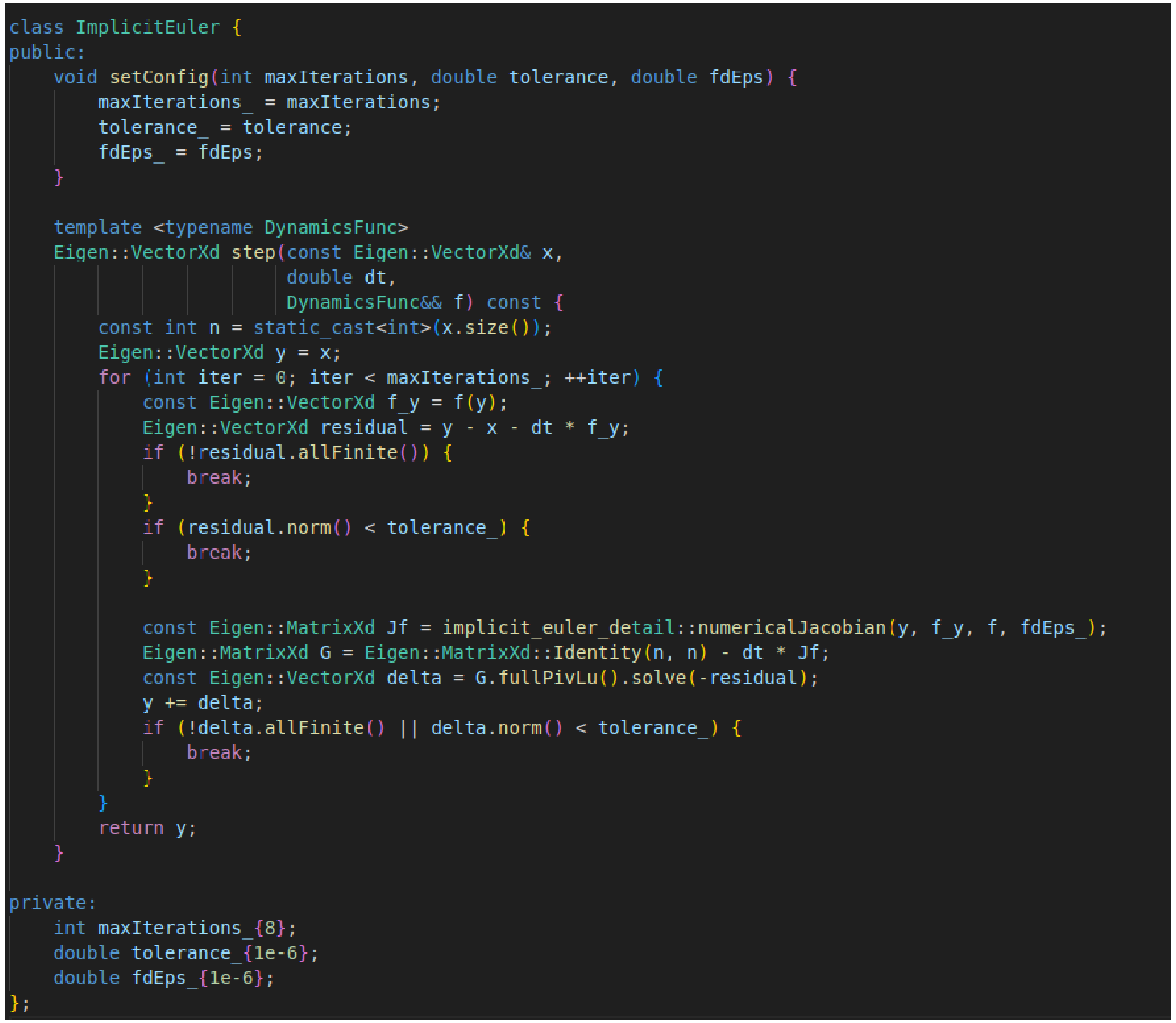
$$\mathbf{X}_{\text{arm}_\text{i}} = \begin{bmatrix} q_{BH} & {}_P\omega_{\text{rel}_{BP}} \end{bmatrix}$$

First page of the derivation of the model.

Current state of the project - integrators

Multiple integrators have been implemented featuring:

- Implicit and explicit Euler
- Implicit and explicit Runge-Kutta 4



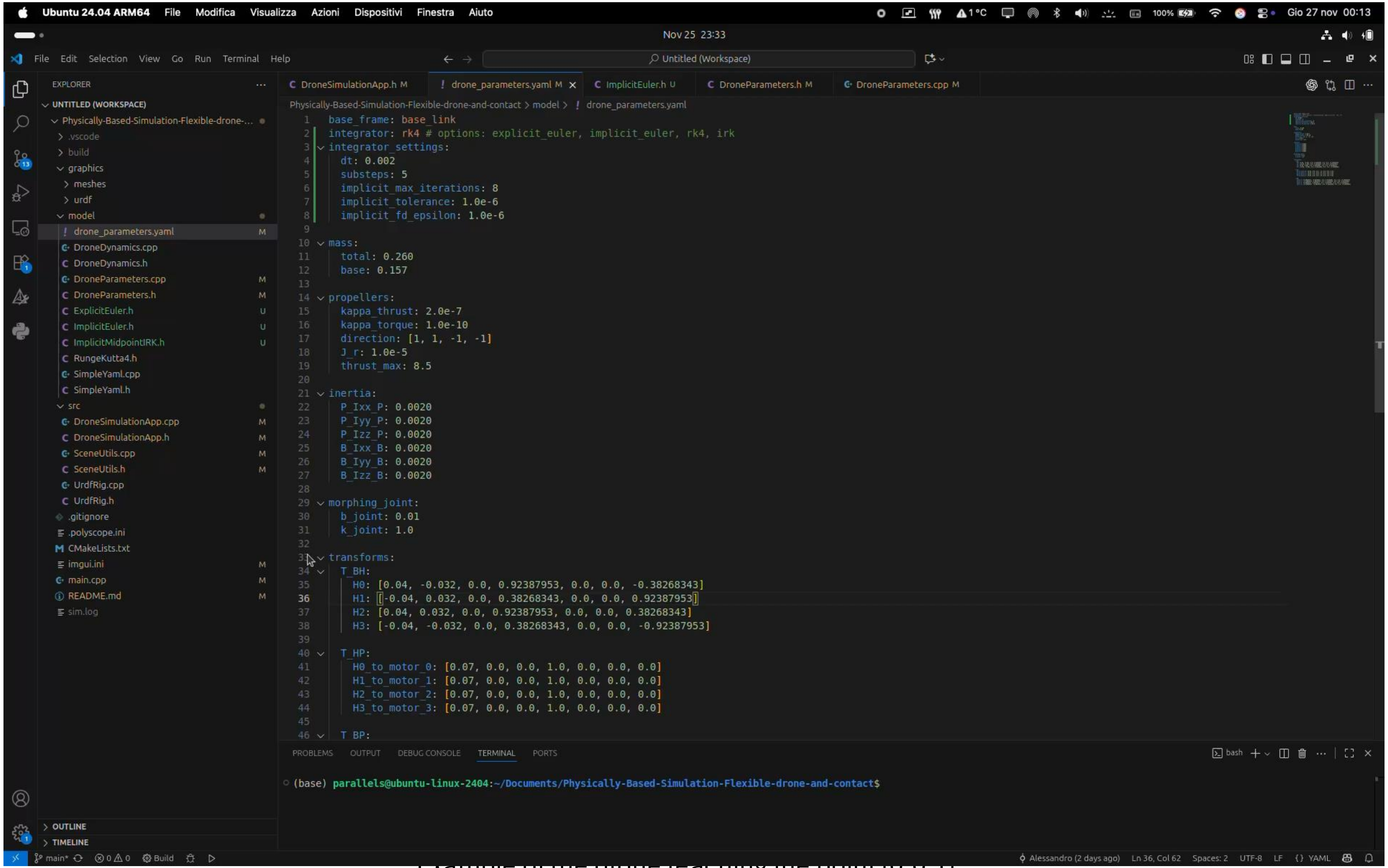
```cpp
class ImplicitEuler {
public:
    void setConfig(int maxIterations, double tolerance, double fdEps) {
        maxIterations_ = maxIterations;
        tolerance_ = tolerance;
        fdEps_ = fdEps;
    }

    template <typename DynamicsFunc>
    Eigen::VectorXd step(const Eigen::VectorXd& x,
                         double dt,
                         DynamicsFunc&& f) const {
        const int n = static_cast<int>(x.size());
        Eigen::VectorXd y = x;
        for (int iter = 0; iter < maxIterations_; ++iter) {
            const Eigen::VectorXd f_y = f(y);
            Eigen::VectorXd residual = y - x - dt * f_y;
            if (!residual.allFinite()) {
                break;
            }
            if (residual.norm() < tolerance_) {
                break;
            }

            const Eigen::MatrixXd Jf = implicit_euler_detail::numericalJacobian(y, f_y, f, fdEps_);
            Eigen::MatrixXd G = Eigen::MatrixXd::Identity(n, n) - dt * Jf;
            const Eigen::VectorXd delta = G.fullPivLu().solve(-residual);
            y += delta;
            if (!delta.allFinite() || delta.norm() < tolerance_) {
                break;
            }
        }
        return y;
    }

private:
    int maxIterations_{8};
    double tolerance_{1e-6};
    double fdEps_{1e-6};
};
```

Implementation example of Implicit Euler

Current state of the project - visualization

The whole simulation stack relies (for now) on polyscope.

A simple PD on the altitude has been implemented to showcase correctness of the system.



```yaml
base_frame: base_link
integrator: rk4 # options: explicit_euler, implicit_euler, rk4, irk
integrator_settings:
  dt: 0.002
  substeps: 5
  implicit_max_iterations: 8
  implicit_tolerance: 1.0e-6
  implicit_fd_epsilon: 1.0e-6

mass:
  total: 0.260
  base: 0.157

propellers:
  kappa_thrust: 2.0e-7
  kappa_torque: 1.0e-10
  direction: [1, 1, -1, -1]
  J_r: 1.0e-5
  thrust_max: 8.5

inertia:
  P_Ixx_P: 0.0020
  P_Iyy_P: 0.0020
  P_Izz_P: 0.0020
  B_Ixx_B: 0.0020
  B_Iyy_B: 0.0020
  B_Izz_B: 0.0020

morphing_joint:
  b_joint: 0.01
  k_joint: 1.0

transforms:
  T_BH:
    H0: [0.04, -0.032, 0.0, 0.92387953, 0.0, 0.0, -0.38268343]
    H1: [-0.04, 0.032, 0.0, 0.38268343, 0.0, 0.0, 0.92387953]
    H2: [0.04, 0.032, 0.0, 0.92387953, 0.0, 0.0, 0.38268343]
    H3: [-0.04, -0.032, 0.0, 0.38268343, 0.0, 0.0, -0.92387953]

  T_HP:
    H0_to_motor_0: [0.07, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
    H1_to_motor_1: [0.07, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
    H2_to_motor_2: [0.07, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
    H3_to_motor_3: [0.07, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]

  T_BP:
```

```
(base) parallels@ubuntu-linux-2404:~/Documents/Physically-Based-Simulation-Flexible-drone-and-contact$
```

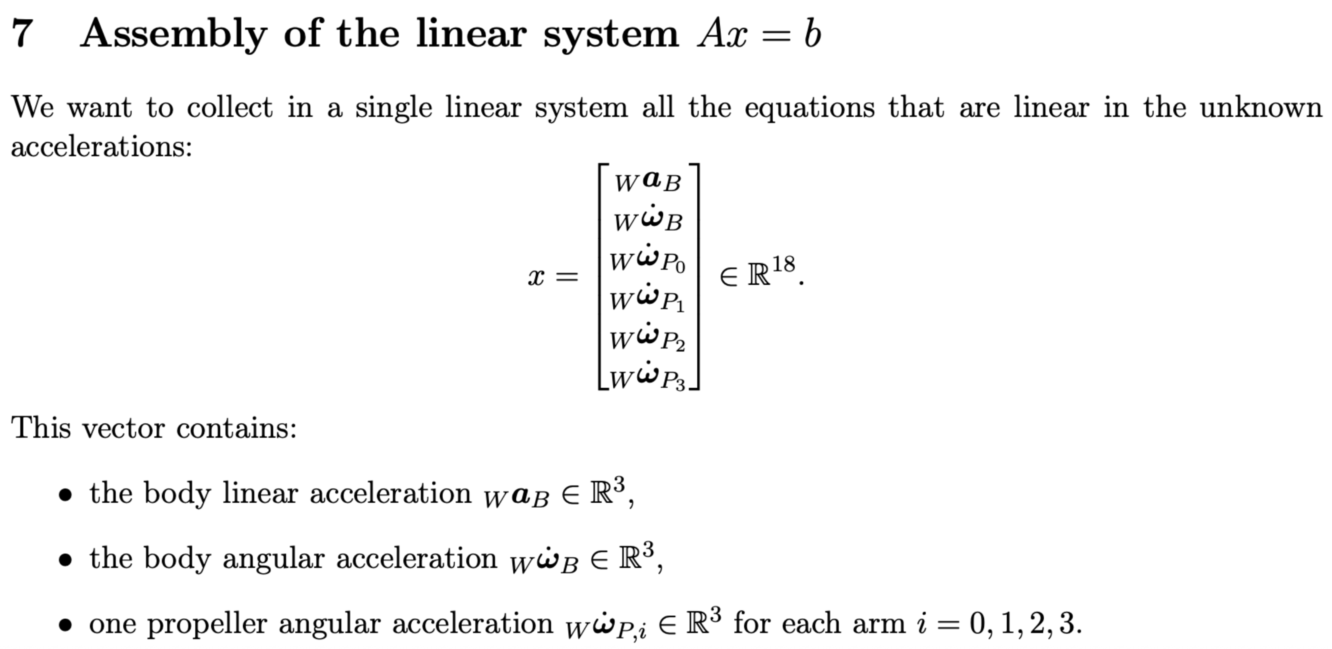
Example of the drone reaching the point (0,0,1)

Encountered Problems – Stiff dynamics and slow integration

Because the system is very nonlinear the whole problem to model was very hard and time consuming.

The coupled equation for the acceleration makes the simulation of the system slow.

Even worse with an implicit integrator (preferred to handle contacts)



7 Assembly of the linear system $Ax = b$

We want to collect in a single linear system all the equations that are linear in the unknown accelerations:

$$x = \begin{bmatrix} {}_W\boldsymbol{a}_B \\ {}_W\dot{\boldsymbol{\omega}}_B \\ {}_W\dot{\boldsymbol{\omega}}_{P_0} \\ {}_W\dot{\boldsymbol{\omega}}_{P_1} \\ {}_W\dot{\boldsymbol{\omega}}_{P_2} \\ {}_W\dot{\boldsymbol{\omega}}_{P_3} \end{bmatrix} \in \mathbb{R}^{18}.$$

This vector contains:

- the body linear acceleration ${}_W\boldsymbol{a}_B \in \mathbb{R}^3$,
- the body angular acceleration ${}_W\dot{\boldsymbol{\omega}}_B \in \mathbb{R}^3$,
- one propeller angular acceleration ${}_W\dot{\boldsymbol{\omega}}_{P,i} \in \mathbb{R}^3$ for each arm $i = 0, 1, 2, 3$.

Linear system 18x18 that needs to be solved

Next Steps – Handling contacts

Until now the whole system is evolving using 5 reference frames, no discretization of the body is needed, also because it behaves as a rigid body (no elasticity modeled).

How to handle contacts need to be though carefully (suggestion are very welcome).

One idea is to try to use the mesh from the STL to do the narrow phase of the contact detection and then try to generalize this to the single body that is colliding (we preview this to be very unstable)

Another approach would be to approximate the system using convex hulls, to try to bypass the mesh, since not “practically” needed.

Once the collision detecting phase has been done, we plan to implement both the impulse based correction and the barrier potential formulation, even if the requirement of the use of the implicit integrator could make everything much slower.

Physically based simulation in Computer Graphics

Milestone presentation Group 11

Alessandro Petitti

Luca Monegaglia

Gennaro Guidone