# Field and Service Robotics
# Homework 5

Riccardo Aliotta P38000143

June 2023

## 1 Buoyancy effect

The Buoyancy effect is a phenomenon caused by the displacement of fluid in which the UUV is (partially or completely) submerged due to its volume; it is an hydrostatic phenomenon, as it only depends on the volume of fluid displaced and its density (it also depends on the gravity acceleration) and not on the actual movement of the UUV. The effect is that, in the buoyancy center of the UUV (which coincides with the geometric center assuming that the fluid has constant density, as it generally is for underwater applications), a force is applied that is equal to the gravitational effect that would be exerted on the displaced amount of fluid, but it is directed upwards instead of downwards. This is caused by the increase in pressure in the fluid as the depth increases, which in turn causes a force thrusting the submerged body upwards (Archimedes's principle). This effect is theoretically present also in aerial robotics applications, but is (in almost all cases) neglected as the density of air is not comparable at all with the UAV's one, generating negligible forces; meanwhile, for UUVs, the densities are generally comparable, and so the buoyancy effect cannot be neglected anymore.

As per its expression, the buoyancy force in the ENU body frame is expressed as

$$f_b^b = -R_b^T \begin{bmatrix} 0 \\ 0 \\ b \end{bmatrix} = -R_b^T \begin{bmatrix} 0 \\ 0 \\ \rho \Delta g \end{bmatrix}$$

where $\rho$ is the fluid density, $\Delta$ is the volume of the UUV and $g$ is the gravity acceleration. It is important to underline that, in case the Center of Mass and the Center of Buoyancy do not lay on the same vertical axis, then a torque is also generated so as the two can stay on the same axis (the torque is so that the CoB is higher than the CoM). If the two coincide, however, this torque is absent for every attitude.

It is useful to see that in UUV the buoyancy and gravitational effects are combined and called Restorative Forces $g_{rb}^b$, which are obtained as

$$g_{rb}^b = -\begin{bmatrix} f_g^b + f_b^b \\ S(r_c^b) f_g^b + S(r_b^b) f_b^b \end{bmatrix}$$

where $r_c^b$ is the position of the Center of Mass and $f_g^b$ is the gravity force contribution

# 2 Underwater Robotics Phenomena

In this section, we will look at some questions about Underwater Robotics Phenomena, providing an answer and a brief explanation.

## 2.1 The Added Mass Effect considers an Additional Load

***False.*** While it is true that the Added Mass Effect causes an increase in the needed force to accelerate the UUV, it is not an actual additional load, but is caused by the surrounding fluid that has to be moved along with the UUV due to fluid viscosity: in fact, this fluid must be accelerated at the same velocity of the UUV itself, increasing the actual force necessary to achieve the same acceleration and thus giving the impression of an increased load. Also, when we model this effect, the resulting added matrix contributions don't share the same properties as true mass ones (specifically, they are at least positive semi-definite, while true mass contributions are always positive definite).

## 2.2 The Added Mass Effect is considered because Water Density is comparable to UUV Density

***True.*** In Underwater Robotics, it's necessary to consider the Added Mass Effect, as the volume of fluid moved has a mass comparable to the UUV's one due to the comparability of the two's densities. On the contrary, in aerial applications the volume of fluid has a negligible mass due to the great difference between the densities, so the force needed to accelerate it is much lower than for underwater applications, and is thus neglected.

## 2.3 The Damping effect helps in the Stability Analysis

***True.*** In the stability proof for the Mixed Earth/Vehicle-fixed-frame-based model-based controller, the inclusion of the damping effect term in the UUV dynamic model leads to the presence in the Lyapunov candidate function's derivative $\dot{V}$ of the positive definite damping matrix $D_{RB}$ in addition to the positive definite matrix $K_D$ defined in the controller; since it appears in a quadratic term which is preceded by a minus sign, the higher its value the more the $\dot{V}$ gets negative, which is helpful as it actually is negative semi-definite and needs Barbalat's Lemma to prove the stability of the controller. We can say that the presence and the entity of the $D_{RB}$ damping matrix can reduce the needed gain $K_D$ to reach stability.

## 2.4 The Ocean Current is considered constant, better if with respect to the body frame

***False.*** While it's true that the Ocean current is, for simplicity, considered constant, as it often is for the periods of time in which the UUV operates and it simplifies the controller, it is however considered constant with respect to the world frame, as in actuality we cannot consider it constant in body frame: if we did that, it would mean that the current varied as fast as the UUV, changing direction as soon as the UUV rotates, and changing absolute speeds as to match the UUV itself, which is absurd.

# 3 Quadruped Robot Simulation Analysis

In this section we will look at the results for the simulation of a quadruped robot, commenting on its behaviour while changing parameters as the mass, the proportional control gain for the swing leg task, the desired forwards velocity, and also different gaits. For some of the simulations (the ones thought to be the most relevant) video recording and plots will be provided, as well as an Excel file containing all the simulated parameter sets and a brief explanation of the results, on the GitHub page `https://github.com/RiAliotta/FSR_Homework5`. The MATLAB code used is from "Representation-Free Model Predictive Control for Dynamic Motions in Quadrupeds", Transactions on Robotics [Yanran Ding, Abhishek Pandala, Chuanzheng Li, Young-Ha Shin, Hae-Won Park], and simulates a quadruped robot system, controlled via the use of an MPC. The quadratic problem solver qpSwift has been employed to resolve the minimization problems while accounting for physical constraints (dynamical consistency and fixed leg contacts as equalities, non-sliding contacts, torque boundedness and leg swing task planning as inequalities). As the code came complete with all functionalities and ease of configuration, the only aspect missing from it to be able to actually use it was the computation of the *zval* variable through the qpSolver, which actually gives the Quadratic Problem's solutions, and allows us to control the quadruped robot.

We will now proceed to show the simulations' results.

## 3.1 Gait Description

By using the default parameters (except for the simulation time, which has been extended to 6 seconds instead of 4.5) a baseline for the subsequent modifications has been established for each gait, and their evolution can be seen in Figures 1 through 6. Before analysing the simulations' results we shall briefly describe how each gait should work.

1. Trot: The robot moves opposite diagonal legs together, and it must also have at least 2 legs touching the ground at all times, so before moving the stance legs, the swing legs must complete the movement and become stance, too.

2. Bound: Moves the front/hind legs together, which will lead to a "Back and Forth bouncing" movement. It could have a flight phase.

3. Pacing: Moves the same side legs together, leading to "Lateral bouncing" movement. There is no flight phase, as the legs either all touch the ground or just two at a time.

4. Gallop: The legs move alternatively, which can lead to having two, one or even zero legs in contact with the floor. This should be a fast gait.

5. TrotRun: Similar scheduling as the trot, however in this case the legs are never all in contact with the ground, but are moved as to get a flight phase. Should be a faster alternative to the trot.

6. Crawl: At most one single leg is swinging at any given time, as to achieve a statically stable gait.

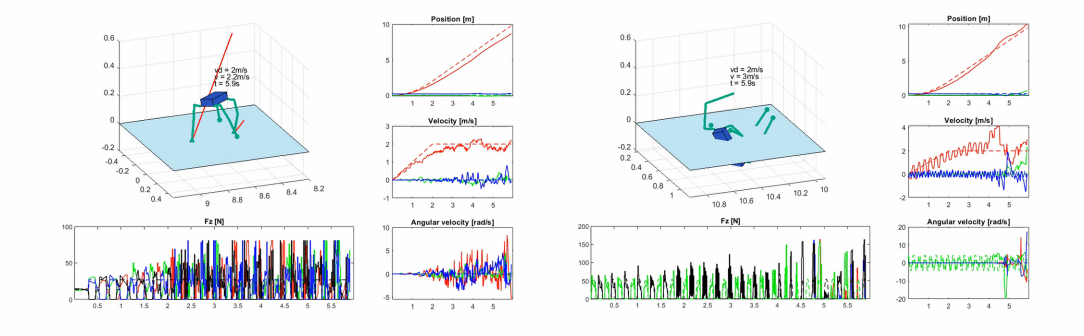


Figure 1: Trot at 2 m/s




Figure 2: Bound at 2 m/s

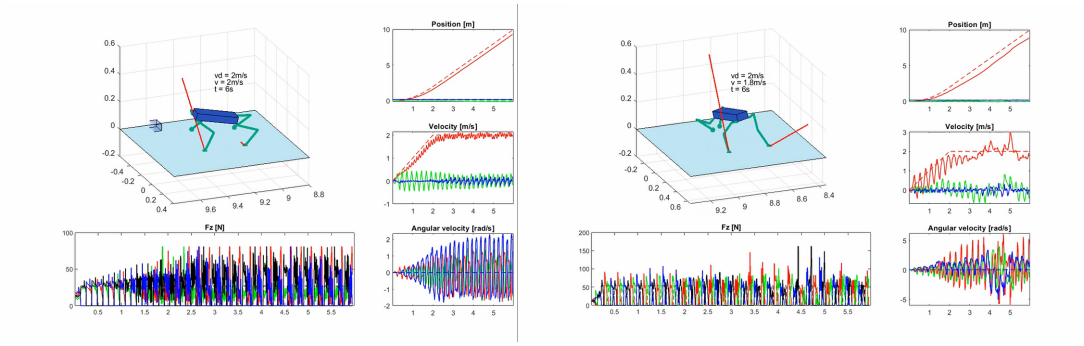


Figure 3: Pacing at 2 m/s




Figure 4: Gallop at 2 m/s

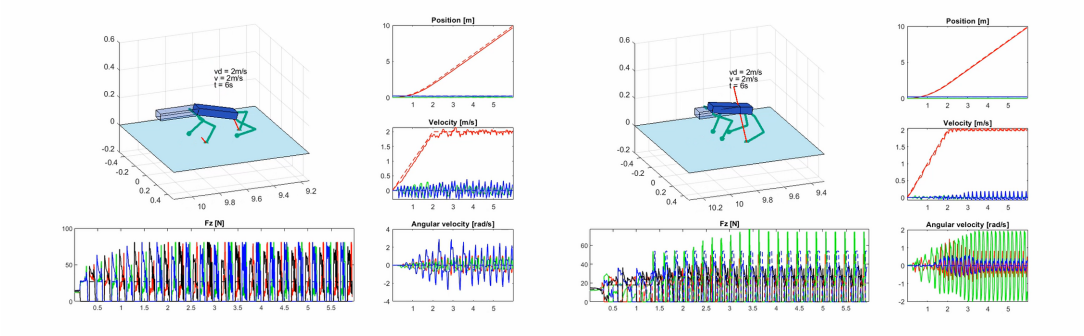


Figure 5: TrotRun at 2 m/s




Figure 6: Crawl at 2 m/s

## 3.2 Default Parameters simulations

It can be seen that a reference velocity of $2m/s$ is not viable for almost all gaits, as Bound, Trot and TrotRun violate physical constraints in simulation (Bound: falls over due to excessive speed; Trot: the legs compenetrate the floor and it assumes an unnatural pose; TrotRun: the legs compenetrate the floor), while the others have bad performances and also undesirable behaviours (Pacing: the hind legs are hazardously close to the floor; Gallop: assumes

an uncomfortable and undesired yaw while moving still in the correct direction; Crawl: the performances are actually good but the gait changes into a gallop). It is also useful to denote that in the simulation only the feet exert forces through contact with the ground, while the rest of the legs (and the body itself) do not impact the floor but compenetrate or even pass through it, without exerting forces; in some cases where the simulation breaks, moreover, the legs may achieve a singular configuration (either fully retracted or extended) before breaking physical constraints (ground compenetration or leg stretching).

### 3.3 Reference Speed with Default Parameters

After this, we have identified a common valid desired velocity of $0.5m/s$ for all the gaits, used from now on as a reference, and from this the parameters have been then varied one at a time. As per the new references, we can comment on how the performances vary with respect to the default parameters.

1. Trot: The legs now have the same height, and the tracking error of the Center of Mass is modest. There is a little velocity oscillation.

2. Bound: Assumes a "bouncing" back and forth behaviour, which overall allows the CoM to track with a modest error the desired position.

3. Pacing: The CoM has a modest tracking error, with a small lateral offset; the lateral oscillation is not in phase with the desired reference.

4. Gallop: The CoM has a greater error with respect to the other gaits (about a body length), great oscillations on the x axis velocity (Back and forth movement), and also has a lateral position error; moreover, there still is an undesired yaw that makes the robot move in the correct direction but skewly.

5. TrotRun: The CoM has similar errors to the simple Trot case, but the hind legs (and so the hind side of the robot) are lower than desired, imposing an undesired pitch to the body. Velocity oscillations are much bigger than Trot along the vertical axis (the flight phase brings a much more "bouncy" behaviour) and much lower along the x axis.

6. Crawl: Works as intended, with a small CoM tracking error and a small pitch oscillation.

By looking at these results, we can notice that a tracking error along the x axis is always present, and this could be attributed to an actual acceleration lower than the desired one or to the absence of an integral action in the implementation of the controller (the MPC control has not been implemented by us so we can't say for sure). However, some gaits have smaller errors than others, and this seems to relate to the "dynamicity" of the gait: the crawl, which is the only statically stable gait tested, is the one with the lowest error, followed by trot and trot run (the flight time is very small in trot run), in which the support line (a possible degeneration of the support polygon for dynamically stable gaits) connects the stance legs, which are positioned diagonally (so the farthest) and so passes through the body's projection on the horizontal plane; pacing and bound have similar errors (bound has a more oscillatory behaviour but the average error is very similar), and the support line in this case is at the edge of the body projection, while the gallop has the worst performances,

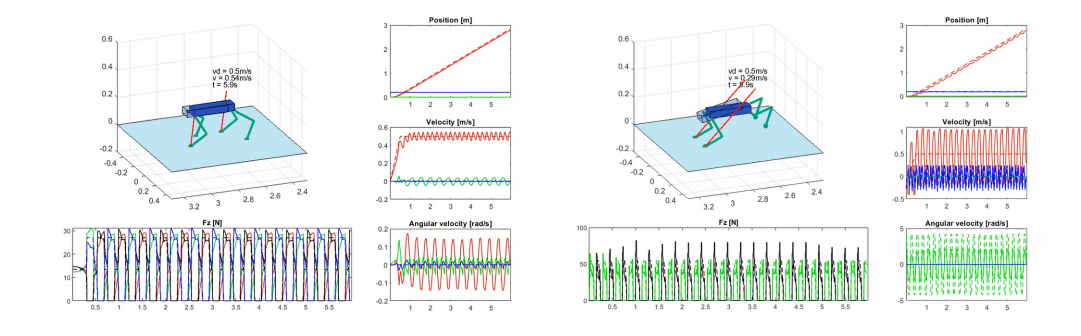


Figure 7: Trot at 0.5 m/s




Figure 8: Bound at 0.5 m/s

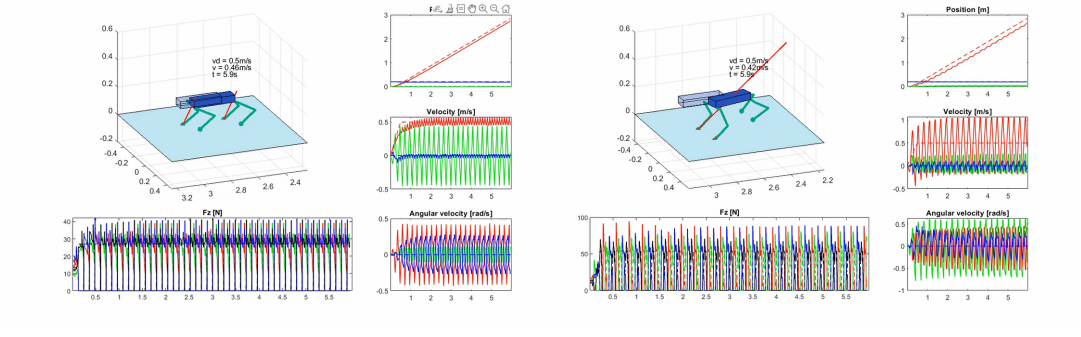


Figure 9: Pacing at 0.5 m/s




Figure 10: Gallop at 0.5 m/s

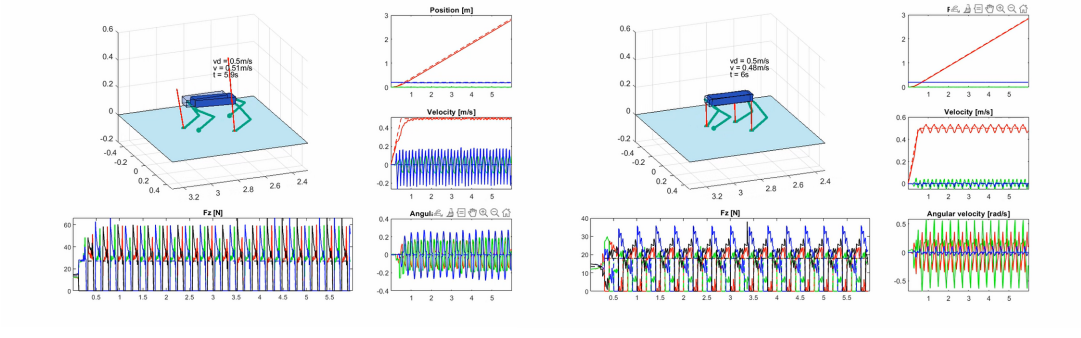


Figure 11: TrotRun at 0.5 m/s




Figure 12: Crawl at 0.5 m/s

and in its case the support polygon degenerates mostly into a single point, for short times into a line, and also flight times are of relevant duration. We can then say that there seems to be a correlation between the position and type of the robot's support polygon during the gait's execution and the entity of the tracking error along the x axis.

## 3.4 Parameter Variations

We now move on to modifying some parameters of the system and check how the performance vary based on that, drawing general conclusions and underlining some particular cases. Not all the simulations done are reported/plotted, but they have been listed and briefly commented, and as already said can be found in the GitHub.

### 3.4.1 Mass Variations

The mass has been both lowered and increased for all the gaits, which actually allows us to make a logical and easy to generalize remark. When the mass is reduces, the errors also get smaller, as do the angular velocities and the contact forces: this is logical, as lower mass means less gravitational force, lower inertia and less energy needed to control it, making it easier to lower the tracking error. Vice versa for increasing the mass. Exception is for the bound gait, in which a decrease in mass causes a decrease in contact forces and in the x axis velocity oscillation amplitudes, but an increase in the x axis errors, while the angular velocities stay the same: this could be attributed to the peculiarity of the bound gait with respect to the other ones. The opposite happens when increasing the mass instead. Moreover, for masses of over 10.5 $kg$, during the pacing gait the quadruped becomes unable to sustain its own weight and falls over. This could be because a solution to the quadratic problem could not be found as to satisfy all the physical constraints and at the same time track correctly the trajectory.

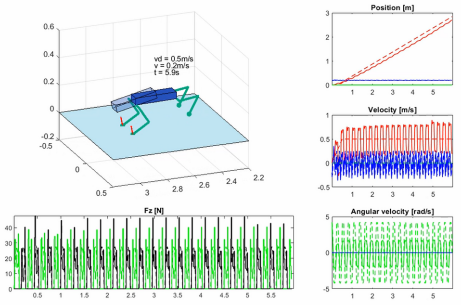


Figure 13: Bound with m = 2.75 kg

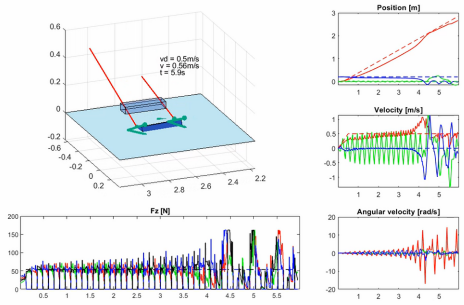


Figure 14: Pacing with m = 11 kg

### 3.4.2 Velocity Variations

From the reference speed of 0.5 $m/s$, the performance of the gaits at also a speed of 1 $m/s$ has been tested, and as imaginable the gaits achieve worse performances: in particular, errors are more dramatic over all gaits, while for the Crawl the actual Crawl stance is lost in favor of a mixed gait, where the front legs and hind legs are moved alternatively, obtaining both a phase in which all legs are stance and a flight phase, with two intermediate two-stance legs' phases (Sequence is roughly Hind - Flight - Front - All, but it can be better visualized in the video on the Github page); in the Gallop, moreover, the yaw error is again present as per the 2 $m/s$ case, even though less dramatic. This results just confirm that this controller is not suitable for all high-speed conditions, as shown in the first part of this section.

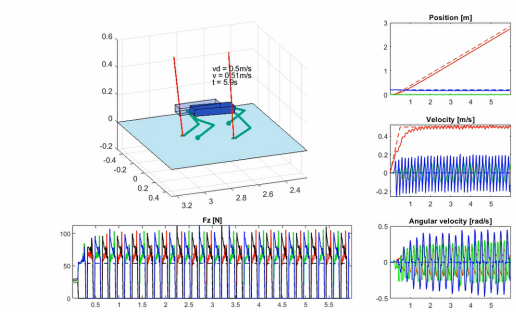


Figure 15: TrotRun with m = 11 kg

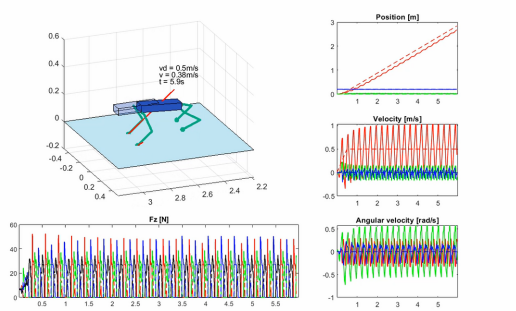


Figure 16: Gallop with m = 2.75 kg

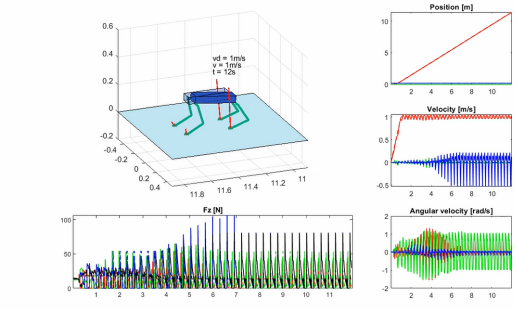


Figure 17: Crawl at 1 m/s

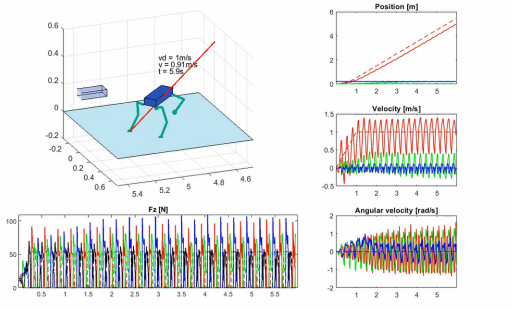


Figure 18: Gallop at 1 m/s

### 3.4.3 KpSwing Variations

Increasing the value for the swing leg gain KpSwing has brought no noticeable variation in any of the measures among all gaits (a gain of 300000 has been tested but apart from requiring 20 minutes of computation time did not have any effect on the performances), instead the reduction of the same gain has different effects on performance, but by looking at the simulations we can say that the general effect on the swing legs' motion planning is to reduce the mobility (or at least the space traveled during the swing phase). While this impacts negatively some gaits' performances, it actually betters the performance of Gallop and Trot Run while still reducing swing leg movement. For the Gallop actually we can see that, while in previous tests it had the tendency to rotate counter-clockwise around the z axis, in this case we have a clockwise rotation instead.

### 3.4.4 Friction Variations

The friction coefficient has been modified, changing it to 0.1 and 0.5 on almost all gaits, achieving mixed results. For the Crawl gait the change in friction had no visible effect, while for the Trot and TrotRun there is only a slight deterioration of performances but overall still pretty close to the reference ones. For what regards the other gaits, however, things are different. The Bound gait has an initial slippage of the legs, as greater as lower the coefficient is, that causes an overshoot in velocity and, after catching up to the reference, in

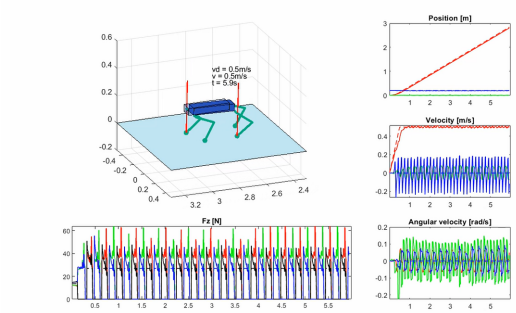


Figure 19: TrotRun for Kp = 30

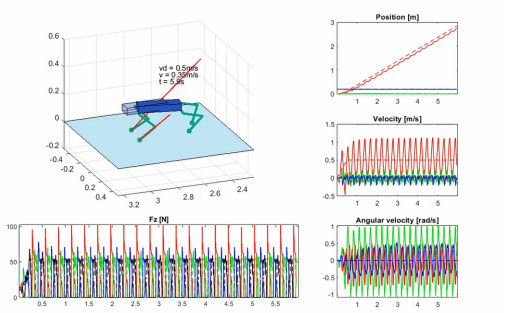


Figure 20: Gallop for Kp = 30

position, which is not recuperated. The Gallop gait has, for a coefficient of 0.5, an initial slippage and then a delay on the tracking, as well as the already seen undesired yaw rotation, this time clockwise. For a lower coefficient of 0.1, instead, the robot loses control and the legs compenetrate the floor, as the simulation again breaks physical constraints. The Pacing gait is, instead, unstable for both values of the coefficient, falling through the ground and breaking again the simulation. Higher values of the coefficient (>0.75) allow it to be stable, with a simple degeneration of performances.

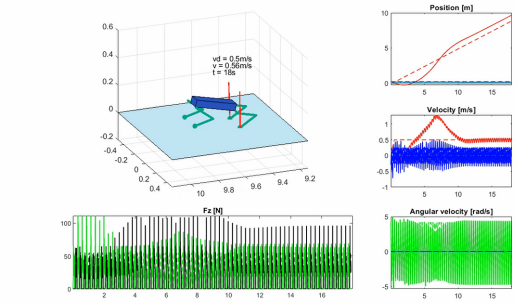


Figure 21: Bound for $\mu = 0.1$

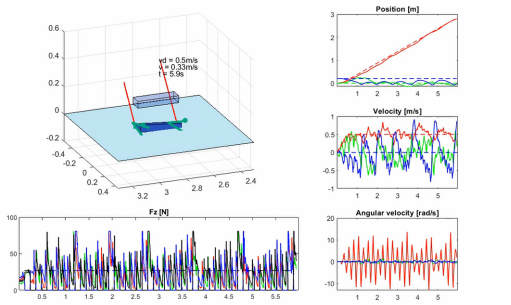


Figure 22: Pacing for $\mu = 0.5$

# 4 Single Legged Robot

In this section we will analyze some problems about the single legged robot in figure 25, where the leg and the foot are considered free of mass, which will only be attributed to the point-like body of mass $m$, with $P$ representing the ankle, which for now will be considered non actuated, $H$ the heel and $T$ the toe; we will answer to three questions regarding the robot, providing a brief reasoning about it.

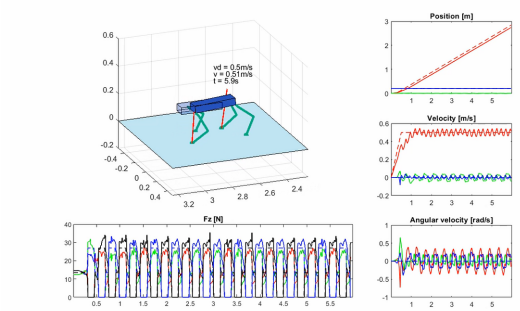


Figure 23: Trot for $\mu = 0.1$

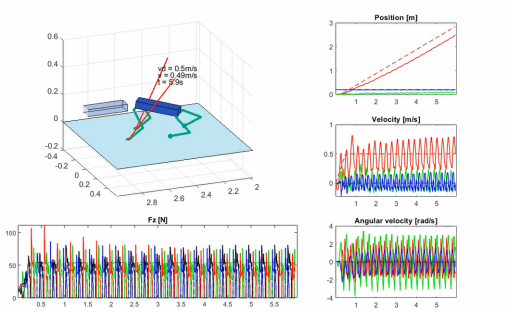


Figure 24: Gallop for $\mu = 0.5$

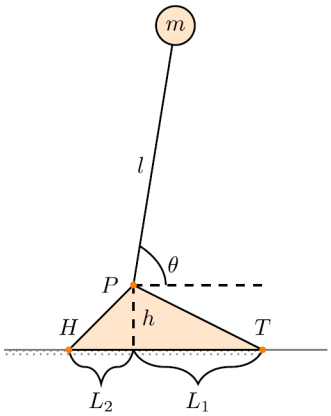


Figure 25: The Single-legged robot in question

## 4.1 Is the system stable for $\theta = \frac{\pi}{2} + \epsilon$?

***No.*** If we assimilate this system to an inverted pendulum (with $\theta = \frac{\pi}{2}$ at the upmots point), for every value of $\theta$ different from exactly $\frac{\pi}{2}$ the mass is bound to fall away towards the ground, so the robot will not be able to stand and will in fact be unstable. We could notice that for small enough values of $l$ (shorter than the minimum between the lengths of $\overline{PH}$ and of $\overline{PT}$) the mass will fall not on the ground but on its own foot, from which it still won't be able to move.

## 4.2 Zero-Moment Point computation

The Zero-Moment Point (or ZMP) can be computed on the basis of geometric parameters and on the Center of Mass motion variables $p_c, \dot{p}_c$ and $\ddot{p}_c$, and it can be defined as

$$p_z^{x,y} = p_c^{x,y} - \frac{p_c^z}{\ddot{p}_c^z - g_0^z}(\ddot{p}_c^{x,y} - g_0^{x,y}) + \frac{1}{m(\ddot{p}_c^z - g_0^z)} S\dot{L}^{x,y}$$

where with the apex $x, y$ is intended a two-dimensional vector referred to, respectively,

the $x$ and $y$ coordinates, $p_c$ is the position of the Center of Mass, $g_0$ is the gravity contribution, $L = I\omega$ is the angular momentum (can be in general assumed $\dot{L} = I\dot{\omega}$) and $S \in SO(2)$ is a matrix

$$S = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

In our case, we have a two-dimensional system, so we can ignore the $p_z^y$ coordinate, and the quantities necessary for the computation are, considering the ankle as zero value for the $x$ coordinate (positive toward the toe) and the floor as zero value for the $z$ value,

$$\begin{aligned} p_c^x &= l\cos\theta \\ p_c^z &= l\sin\theta + h \\ \ddot{p}_c^x &= -l\dot{\theta}^2\cos\theta - l\ddot{\theta}\sin\theta \\ \ddot{p}_c^z &= -l\dot{\theta}^2\sin\theta + l\ddot{\theta}\cos\theta \\ I &= ml^2, \omega = \dot{\theta} \\ \dot{L}^y &= ml^2\ddot{\theta} \\ g_0^x &= 0 \\ g_0^z &= -g \end{aligned}$$

which gives us the final expression

$$p_z^x = l\cos\theta - \frac{l\sin\theta + h}{-l\dot{\theta}^2\sin\theta + l\ddot{\theta}\cos\theta + g}(-l\dot{\theta}^2\cos\theta - l\ddot{\theta}\sin\theta) - \frac{1}{m(-l\dot{\theta}^2\sin\theta + l\ddot{\theta}\cos\theta + g)}ml^2\ddot{\theta}$$

In static cases, so when $\dot{\theta} = \ddot{\theta} = 0$, we can greatly simplify this expression , which becomes $p_z^x = l\cos\theta$, which coincides with the projection of the Center of Mass on the ground.

### 4.3 Non-falling values of $\theta$ assuming an actuated ankle

In order for the robot not to fall, and having assumed that the actuator on the ankle is capable of perfect cancellation of the gravity effects, we shall keep the projection of the Center of Mass inside the support polygon, so we must constrain $l\cos\theta$ to stay between the heel and the toe, and so we should assume $-L_2 < l\cos\theta < L_1$. To satisfy these constraints, we must have $l\cos\theta > -L_2$ and $l\cos\theta < L_1$, which in turn will give us $\arccos\frac{-L_2}{l} > \theta > \arccos\frac{L_1}{l}$, which is the interval of values $\theta$ can assume without the robot falling over.