Ecole Polytechnique Fédérale de Lausanne
Legged Robots
MICRO-507

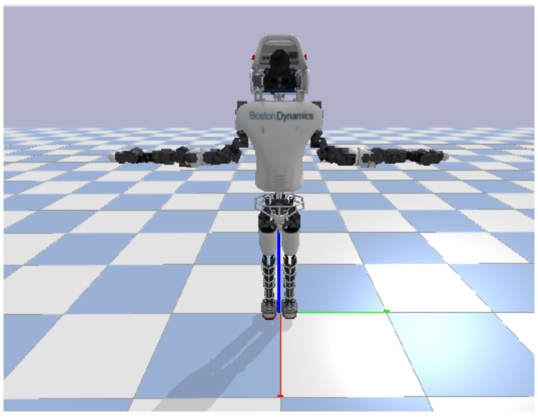
# Locomotion planning based on Divergent Component of Motion (DCM)

*Legged Robots - Project 1*

Catarina Pires   Dimitri Hollosi   Richard Gao

Supervising Teacher:

Auke Ijspeert



Lausanne, November 2021

# Table of Contents

# 1. Introduction

It is required to plan the Center of Mass (CoM) trajectory for a biped using the Divergent Component of Motion (DCM) concept for locomotion on flat terrain. To do this, the method presented in [1] is followed to ultimately aim at providing locomotion for an Atlas robot in a pybullet environment. By generating the CoM motion and coupling this with the Feet Trajectory, the desired joint angles of the walker can be controlled through inverse kinematics formulation.

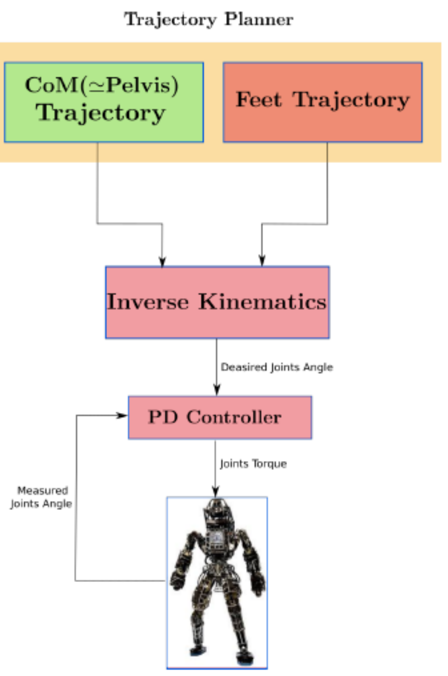


**Figure 1.1:** DCM Trajectory Planner

## 1.1 Theoretical Insights

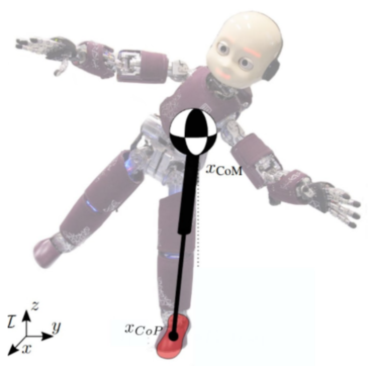


**Figure 1.2:** Inverted Pendulum dynamics - CoM definition

It is desired to find the DCM trajectory from a given constant Center of Pressure (CoP). To this effect, it is necessary to recall the inverted pendulum equation (schematically represented in figure 1.2), as

$$\ddot{x}_c = \omega^2(x_c - r_{CoP}) \tag{1.1}$$

with $x_c$ the CoM position and $\omega = \sqrt{\frac{g}{z_c}}$ is the natural frequency of the DCM dynamics. The solution to equation 1.1 is comprised of a convergent and divergent term (A and B, respectively) as time tends to infinity, as shown in equation 1.2.

$$
\begin{aligned}
x_c(t) &= Ae^{-\omega t} + Be^{\omega t} + x_{base} \\
A &= (-\frac{\dot{x_c}(0)}{\omega} + x_c(0) - r_{CoP})/2 \\
B &= (\frac{\dot{x_c}(0)}{\omega} + x_c(0) - r_{CoP})/2
\end{aligned}
\tag{1.2}
$$

This implies that the term B must be equated to zero in order to capture the unstable part of the dynamics. This in turn allows the more general introduction of the DCM (or instantaneous capture point) which aims to bring the CoM to a desired position, i.e

$$
\xi = x_c + \frac{\dot{x_c}}{\omega} \tag{1.3}
$$

The stable and unstable dynamics (for the CoM and DCM, respectively) can now be derived by mathematical manipulation of the above presented terms as shown in equation 1.4.

$$
\begin{aligned}
\dot{x_c} &= \omega(\xi - x_c) \\
\dot{\xi} &= \omega(\xi - r_{CoP})
\end{aligned}
\tag{1.4}
$$

Assuming the CoP to be of constant value during the single support phase, the DCM trajectory is given by the following solution of the DCM Dynamics (equation 1.5) where T is defined at the step duration.

$$
\begin{gathered}
\xi(t) = r_{cop} + (\xi_0 - r_{cop})e^{i\omega t} \\
t \in [0, T]
\end{gathered}
\tag{1.5}
$$

As a direct consequence of the relation 1.4 shown above, the DCM is constrained (or "pushed") by the CoP and, in turn, the CoM aims to follow the DCM trajectory with its stable dynamics. This is visually represented in figure 1.3, where it is clear that the last DCM position is captured by the CoP at the end of every step.

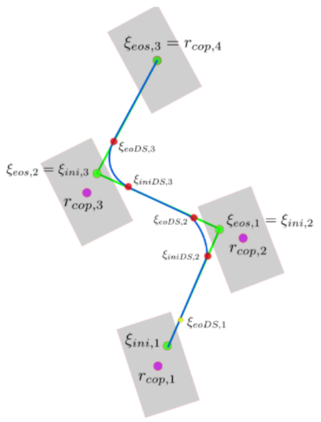


**Figure 1.3:** DCM Trajectory Planner

## 1.2 Report Outline

Section two of this report will cover the methods used to calculate the DCM and CoM trajectories for the Atlas robot. This section also introduces the comparison metric used, Cost of Transport, to compare the Atlas robot and a simulation of a three link biped seen previously.

The third section presents in detail the results of the simulation including plots of the DCM and CoM trajectories. A description of the tuning parameters and their effects on these results is provided along with a qualitative assessment of the robot's performance based on the comparison metric seen previously.

Section four provides further discussion of the obtained results based around the questions asked in the notebook project file. Final conclusions and thoughts are presented in section five.

# 2. Methods

## 2.1 Implementation Strategy

Implementation of the biped simulation relies on three code blocks: Foot Trajectory Planner, DCM Planner and Inverse Kinematics. The Foot Trajectory Planner module is used to provide the foot step positions and other necessary variables such as step length, step duration, pelvis height etc. for the DCM Planner. The output of the DCM Planner module is the vector of DCM and CoM which is used in Inverse Kinematics to model the robot. From the provided methodology in the project assignment [1], only the DCM Planner module *DCMTrajectoryGenerator.py* required implementation (i.e all other libraries/modules and main notebook were also provided in the assignment). The methodology is recalled to be the following [1] :

1. Select the foot step position and step duration based on the desired velocity and considering the kinematic and dynamic constraints of the robot.
2. Place the desired CoP in a fixed location inside of the foot print. This condition guarantees dynamic balance during locomotion.
3. Place the last DCM position on the last CoP (Capturability constraint). For planning, we assume that the DCM will come to a stop over the final previewed foot position.
4. Find the desired DCM locations at the end of each step via recursion using equation 1.5 (from constant desired CoP positions for each step and the last DCM Position, located on the CoP.
5. Compute the referenced trajectories for DCM position (for single support phase) of the *ith* step (based on equs. 1.5 and its recursive counterpart.
6. Compute the boundary conditions of the DCM during double support phase (beginning and end of double support)
7. Interpolate the new DCM values during double support and replace the corresponding single support DCM values
8. Calculate the CoM trajectory by calculating the CoM velocity using equation 1.4 and performing numerical integration to find the positions.

## 2.2 Trajectory Planning

**Listing 2.1:** Implementation of Planning DCM and CoM Trajectories

```
CoPOffset=np.array([0.0,0.0]) #Offset between CoP and footprint position(Ankle position)

DCMPlanner = DCMTrajectoryGenerator(pelvisHeight, stepDuration, ...
    doubleSupportDuration)#We create an object of foot DCMTrajectoryGenerator Class
CoPPositions=np.empty((DCMPlanner.numberOfSteps+1, 3))#Initialization of the CoP array

#In the following we define the foot step positions
for i in range(0,DCMPlanner.numberOfSteps+1):
    if(i%2!=0):
        CoPPositions[i][:]=[(i)*stepLength-CoPOffset[0],stepWidth-CoPOffset[1],0.0]
        if(i==1):
            CoPPositions[i][:]=[(i)*stepLength,stepWidth-CoPOffset[1],0.0]
    else:
        CoPPositions[i][:]=[(i)*stepLength-CoPOffset[0],-stepWidth+CoPOffset[1],0.0]
        if(i==0):
            CoPPositions[i][:]=[(i)*stepLength,-stepWidth+CoPOffset[1],0.0]

DCMPlanner.setCoP(CoPPositions)#We set the desired CoP positions
```

```
18 DCMPlanner.setFootPrints(FootPrints)#We set the foot steps positions
19 DCMTrajectory = DCMPlanner.getDCMTrajectory()#At the end of DCM Planning the size of ...
       DCM vector should be 4320
20 initialCoM = np.array([0.0,0.0,DCMPlanner.CoMHeight])
21 comTrajectory = DCMPlanner.getCoMTrajectory(initialCoM)
22 DCMPlanner.calculateCoPTrajectory()
```

The *DCMTrajectoryGenerator* class contains the methods *getDCMTrajectory* and *getCoMTrajectory* whose outputs are the main objective of the trajectory planning section. The CoP and foot step positions are used to calculate DCM and CoM trajectories and they are defined in the main notebook and set using the *setCoP* and *setFootPrints* methods (lines 17-18 of Listing 2.1). The global variables *pelvisHeight*, *stepDuration* and *doubleSupportDuration* are used to initialise their respective class variables (N.B it is assumed that the pelvis and CoM are at the same point).

Calculation of the DCM trajectory is done in four steps:

1. Find the final DCM position for each step
2. Plan the DCM trajectory during single support
3. Find the boundary conditions of the DCM during double support
4. Find the DCM trajectory during double support and integrate with the single support trajectories

For step one the capturability constraint is defined so that the final DCM position is placed over the final CoP. Now using the constant desired CoP positions and the capturability constraint, the DCM locations for the end of each step can be found via recursion. The implementation of this step is a simple backwards iteration of equation 1.5 and the DCM positions are stored in a vector having the same shape as the CoP.

Knowing the DCM position for the end of each step, the DCM trajectory of the $i$th step can be calculated as the DCM positions at internal step time $t$. Note that $t$ is reset at the beginning of each step. A modified version of equation 1.5 is used to compute this:

$$\begin{aligned} \xi_i(t) &= r_{cop,i} + (\xi_{eos,i} - r_{cop,i})e^{\omega(t-T)} \\ & t \in [0, T] \end{aligned} \tag{2.1}$$

In the code this equation is iterated over the number of sample points for the whole time of walking. The step number is found by floor dividing the corresponding control cycle time by the step duration. The internal step time is found by subtracting the time at the beginning of the corresponding step from the corresponding control cycle time. The results of each iteration are added to a preliminary vector containing only DCM positions during single support.

Using only the single support DCM trajectory leads to discontinuity in the commanded joint torques which is infeasible for a physical robot. This motivates the use of a third-order polynomial interpolation to "round" the edges of the preliminary DCM trajectory corresponding to a smooth transition during a double support phase. Given a desired double support duration $t$, the DCM positions at the beginning and end of double support can be calculated. These boundary conditions are given by the following equations:

$$\xi_{iniDS,i} = r_{cop,i-1} + (\xi_{ini,i} - r_{cop,i-1})e^{-\omega\alpha_{DS,ini}t_{DS}} \tag{2.2}$$

$$\xi_{eoDS,i} = r_{cop,i} + (\xi_{ini} - r_{cop,i})e^{\omega(1-\alpha_{DS,ini})t_{DS}} \tag{2.3}$$

where *iniDS* and *eoDS* are the initial double support and end of double support respectively.

These equations are iterated in the code for each step $i$ to give the double support boundary conditions for each step. The velocities are then found by differentiation as in equation 1.4. Note that the first step is computed differently giving:

$$\xi_{iniDS,1} = \xi_{ini,i} \tag{2.4}$$

$$\xi_{eoDS,1} = r_{cop,1} + (\xi_{ini,1} - r_{cop,1})e^{\omega(1-\alpha_{DS,ini})t_{DS}} \tag{2.5}$$

To perform the cubic interpolation, a polynomial parameter matrix can be found using the double support position and velocity boundary conditions as follows:

$$P = \begin{bmatrix} 2/T_{DS}^3 & 1/T_{DS}^2 & -2/T_{DS}^3 & 1/T_{DS}^2 \\ -3/T_{DS}^2 & -2/T_{DS}^2 & 3/T_{DS}^2 & -1/T_{DS} \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \xi_{iniDS,i} \\ \dot{\xi}_{iniDS,i} \\ \xi_{eoDS,i} \\ \dot{\xi}_{eoDS,i} \end{bmatrix} \tag{2.6}$$

The four elements of the $P$ matrix are manually calculated in the code and this operation is iterated for each step. Now using the parameter matrix, the DCM position and velocity at a time $t$ during double support can be found by interpolation as follows:

$$\begin{bmatrix} \xi(t) \\ \dot{\xi}(t) \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \\ 3t^2 & 2t & 1 & 0 \end{bmatrix} P \tag{2.7}$$

This is iterated in the code for each step and each $t$ during double support. Finally the DCM positions during double support are used to replace the corresponding DCM positions in the preliminary DCM vector giving the final vector of DCM trajectory for the Atlas robot.

Using the completed DCM trajectory vector, the CoM trajectory can be found by substituting the current DCM and CoM values into equation 1.4 to find the CoM velocity and then performing a simple numerical integration to find the CoM position for each sample period used in the simulation.

## 2.3 Comparison metric

It is desired to compare the Cost of Transport (CoT) between the one generated by the Atlas robot and the 3-link Walker designed on Matlab. It should however be noted that the CoT in the previous assignment was described as the effort over a certain displacement, i.e

$$\begin{gathered} CoT = \frac{Effort}{x_{hip}(T) - x_{hip}(1)} \\ Effort = \frac{1}{2TU_{max}^2} \sum_{i}^{T} (u_1(i)^2 + u_2(i)^2) \end{gathered} \tag{2.8}$$

However, a more standard definition of the Cost of Transport is given in equation 2.9. This highlights the energy efficiency from an input power to cover a specific displacement of an object of mass $m$ whose CoM is travelling at a velocity $v$.

$$CoT = \frac{P}{mgv} \tag{2.9}$$

For comparative purposes, a human achieves the lowest CoT of approximately 0.4 when walking at about 6 kilometres per hour, at which speed a person of 70 kilograms has a metabolic rate of around 450 watts [3]. However, considering a legged robot which is fully controllable, the CoT can be operated at a desired efficiency to achieve specific performance. This is due to the fact that the total power is given by the sum of the joint torques multiplied by their angular velocities. A legged robot operated at a high power with a low CoM velocity would thus result in a high Cost of Transport (i.e it is overdriven). However, a low CoT implies an efficient locomotion method as the speed would be higher from a inputted power, making good use of body dynamics.

# 3. Results & Discussion

A qualitative assessment of the simulated Atlas robot performance will be outlined in this section. As will be detailed, the initial parameters for the feet trajectory computation resulted in unstable dynamics. The parameters were therefore iteratively tuned until satisfactory performance was obtained, which was decided to be assessed through the Cost of Transport ratio. Further insights will be brought to the latter through comparative measures between the Atlas and a 3-link biped from previous lab assignments.

## 3.1 First Results and Parameter Tuning

Once the methods described have been implemented, the results obtained for a set of default parameters (table 3.4) are shown in figure 3.1.

**Table 3.1:** Default Parameters

| Default Parameters | |
|---|---|
| Double Support Duration | 0.25 |
| Step Duration | 1.2 |
| Hip height | 0.7 |
| Maximum Foot Height | 0.07 |
| Step Width | 0.12 |
| Step Length | 0.1 |

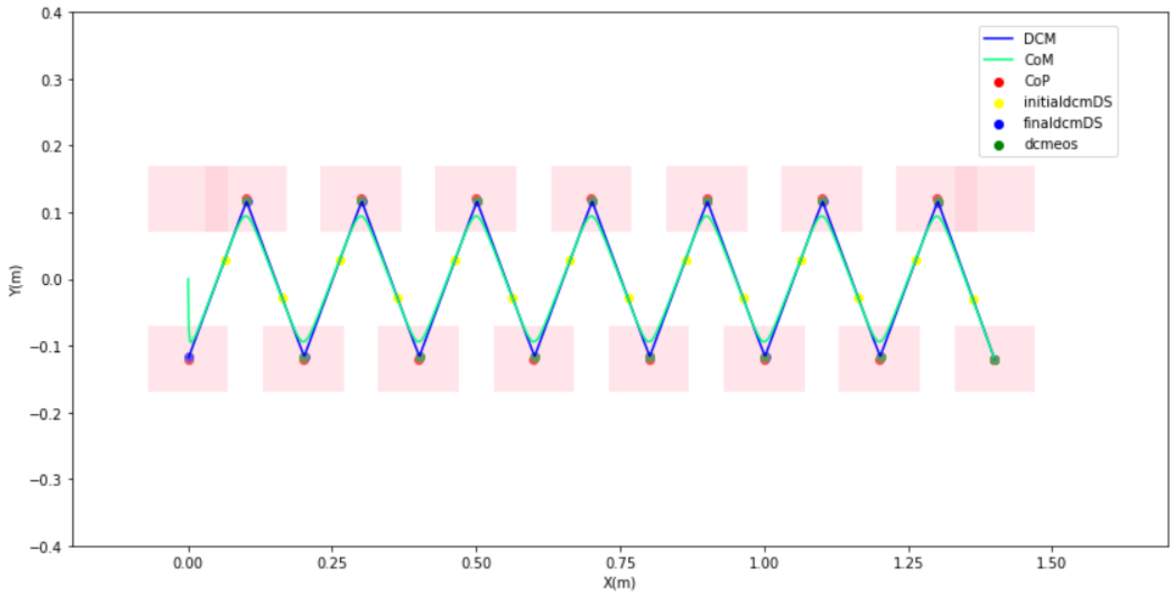


**Figure 3.1:** DCM (single and double support), CoM and CoP trajectories for default parameters

The obtained results showed that the robot did not maintain a stable locomotion, falling sideways before it could take a second step. The robot's loss of balance is due to the fact that the CoM trajectory, with the default parameters, comes too close to the CoP for each step and overlaps with the footstep area (shown in Figure 3.1). Therefore, the parameters were tuned until a stable gait was achieved. This goal was reached solely by changing the step duration from 1.2 seconds to 0.8 seconds, resulting in the DCM, CoM and CoP trajectories presented in figure 3.2. In contrast to the CoM trajectory observed in figure 3.1, the new CoM trajectory never falls within the robot's footprint, hence allowing it to maintain stable locomotion.

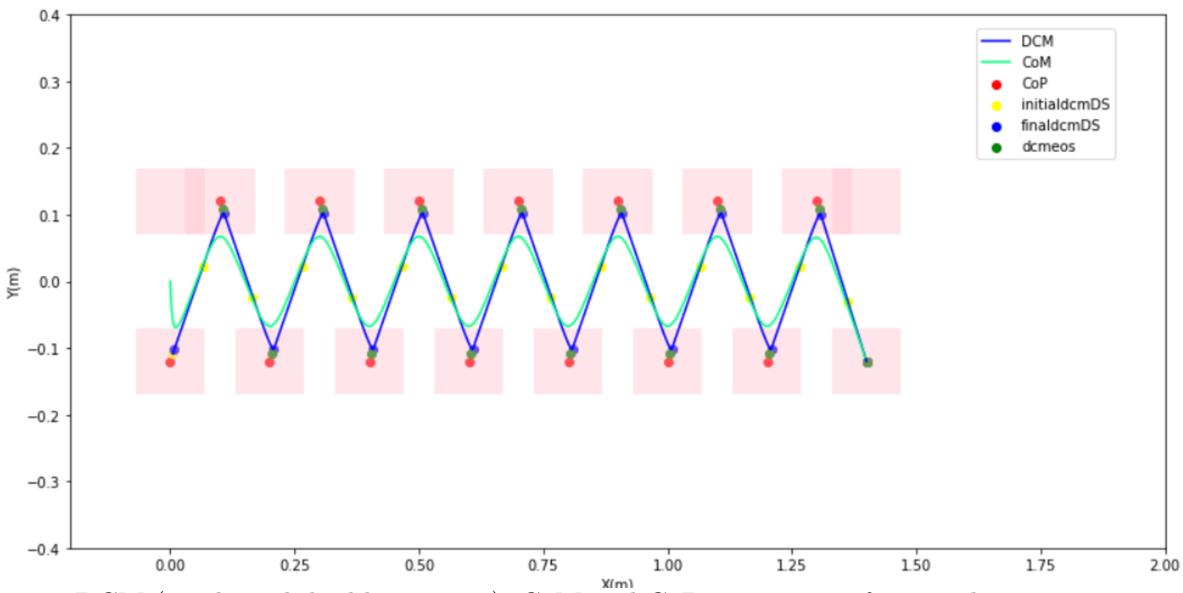


**Figure 3.2:** DCM (single and double support), CoM and CoP trajectories for tuned prameters parameters

## 3.2 Atlas Performance Assessment

### 3.2.1 Comparison with the 3-linked biped

As explained in section 2.3, it is desired to CoMpare the Atlas robot's performance with the 3-link biped from previous assignments. This is done in order to obtain a qualitative performance metric for it.

**Table 3.2:** Atlas and Biped robots Comparison

| Parameter | Atlas | 3-Link Biped |
|---|---|---|
| Mass [kg] | 89 | 31 |
| Power [Watts] | 80.5 | 26 |
| P/mg ratio | 0.09 | 0.09 |
| CoM velocity [m/s] | 0.11 | 0.77 |
| COT | 0.79 | 0.13 |

As shown in table 3.2, the COT for the biped is 0.15 (with a final CoM velocity of 0.77 [m/s]) compared to 0.79 for Atlas (travelling at 0.11 [m/s]). This implies that the biped is more energy efficient than the Atlas as it yields a lower COT. However, this is mainly due to considerable speed difference between the two robots. Indeed, comparison the ratio of the physical parameters of the two robots (i.e, P/mg) shows that they would indeed have similar COT performance if they had similar CoM velocities. This naturally leads to wondering whether the COT of the Atlas robot could be improved, potentially to be as low as the biped's.

### 3.2.2 Parameter Tuning - COT "optimisation"

From the previous comparisons shown here-above, it is desired to increase the speed of the Atlas to that of the biped's. Table 3.3 shows the iterative tuning procedure of the two parameters which were thought to have the strongest influence on the CoM velocity (namely the Step Duration and Step Length).

**Table 3.3:** Performance with Parameter Tuning

| Configuration | Step Duration | Step Length | CoM Velocity [m/s] | **COT** |
|---|---|---|---|---|
| 1 | 1.2 | 0.12 | 0.07 | **1.18** |
| 2 | 0.8 | 0.12 | 0.11 | **0.79** |
| 3 | 0.6 | 0.28 | 0.43 | **0.21** |
| 4 (**Unstable**) | 0.5 | 0.35 | 0.65 | **0.15** |

The CoM velocity progressively increases as the COT decreases. The fourth iteration seemed promising in terms of COT until the Atlas simulation showed unstable dynamics and collapsed. Investigation of the DCM, CoM and CoP trajectories clearly illustrate this behaviour (see figure 3.3). Indeed, the uncompensated increase in velocity results in final DCM points outside the constrained polygon, implying an unstable locomotion as the DCM is not captured by the COP at the end of each step.

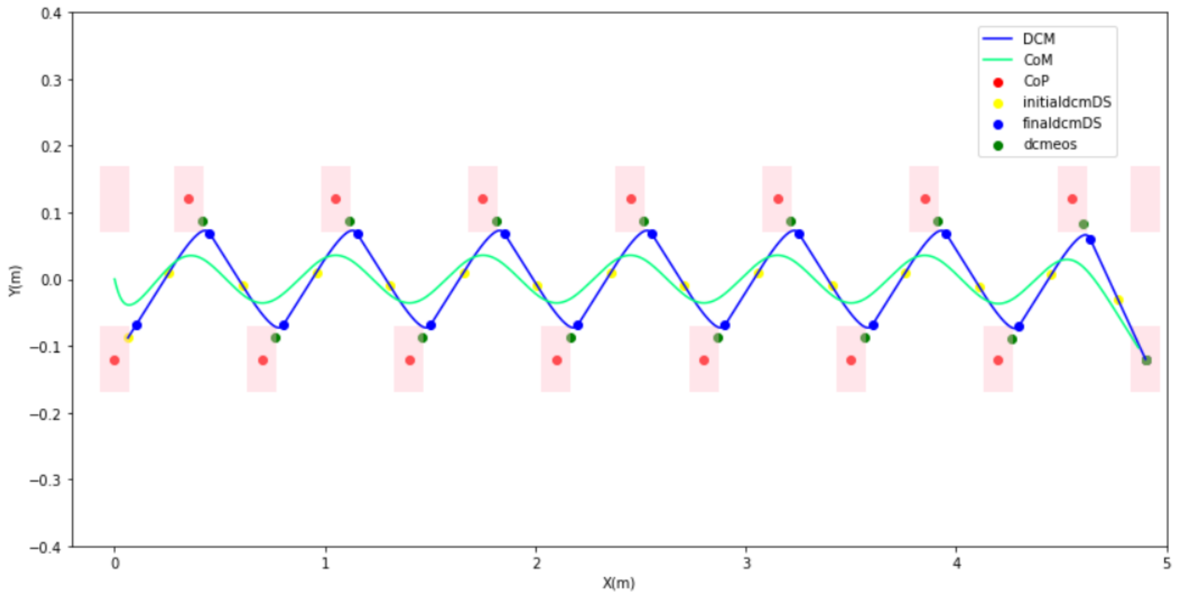


**Figure 3.3:** DCM (single and double support), CoM and CoP trajectories for

### 3.2.3 Final Performance

Further tuning of the parameters allowed to obtain a satisfactory COT of 0.15, albeit with a slight decrease in CoM velocity (from 0.65 to 0.62 [m/s]). Despite not yielding the exact same performance as the biped robot, it can be stated that the increase in velocity allowed to minimise the overall Cost of Transport of the Atlas robot, ultimately resulting in a more satisfactory performance than the initial COT of 0.79.

**Table 3.4:** Final Parameters & Performance

| Final Parameters | |
|---|---|
| Double Support Duration | 0.2 |
| Step Duration | 0.6 |
| Hip height | 0.7 |
| Maximum Foot Height | 0.07 |
| Step Width | 0.12 |
| Step Length | 0.4 |
| **CoM Velocity [m/s]** | **0.62** |
| **COT** | **0.15** |

# 4. Questions

## 4.1 Based on equation 1.5 which physical parameters will affect the rate of divergence of DCM dynamics?

The CoP position and the hip height as they influence the natural frequency of the DCM dynamics.

## 4.2 In the DCM motion planning how do we guarantee to have a Dynamic Balancing?

Dynamic balancing is ensured by placing the desired CoP position in a fixed location inside of the footprint. By calculating the DCM and hence the CoM trajectories using the fixed CoP positions, it is ensured that the CoM never exceeds the CoP and dynamic balancing is achieved.

## 4.3 If we garantee to have dynamic balancing why the robot is not able to walk without parameter tuning?

Despite the CoM being within the limits of the CoP positions, the robot failed to maintain stable locomotion with the default parameters given in table 3.4. It was found through iteration that in order for the robot to maintain stable locomotion, the CoM must not overlap the footprint areas and this is achieved through parameter tuning.

## 4.4 Which parameters you found useful for tunning to have a stable locomotion and whats the value of those parameters?

Given the initial set of parameters the robot did not have a stable locomotion. In testing it was found that doubleSupportDuration, stepDuration, pelvisHeight and stepLength all had a significant impact on the stability of the robot. Of these parameters, pelvisHeight is the only one that affects the robot's posture while the others affect the robot's movement speed. It was found that increasing pelvisHeight from 0.7 to 0.85 gave the robot a stable locomotion however, this resulted in the robot's legs being fully straightened throughout its movement risking damage to the joints. It was decided that, to avoid this, the initial pelvisHeight should be maintained. Therefore, in order to mimic the desired stride given by the initial parameters as best as possible, the stepDuration was changed from 1.2 to 0.8 with all other parameters remaining the same.

## 4.5 Find the cost of transport for this walking planner and do a comparison with the previous matlab homework? What are the source of difference?

See section 3.2 Comparison with the 3-linked biped.

## 4.6 Change the step position and duration in "Planning Feet Trajectories" section, to have a faster locomotion. what's the fastest walking speed that you have achived and what are the corresponding parameters?

The fastest walking speed achieved was 0.62 m/s with the corresponding parameters laid out in table 3.4.

# 5. Conclusive Remarks

This report has presented the method and results of locomotion planning in an Atlas robot based on the Divergent Component of Motion concept presented in [1]. Planning of the DCM and CoM trajectories of the robot was achieved iteratively in python and using fixed CoP and footprint positions as described in section 2. Validation of the *DCMTrajectoryGenerator* class was found by simulating the robot walking in the pybullet environment and which is controlled by the inverse kinematics solved using the CoM and DCM trajectories. As seen in section 3.1 the robot requires parameter tuning of the step position, step duration and CoM height in order to maintain stable locomotion. It was found that keeping the CoM within the CoP was not enough to maintain stability and that ideally the CoM trajectory should not overlap the footprint area. The Cost of Transport metric was used to assess the performance of the model on its energy efficiency in motion. This metric was used to determine the final parameters for the lowest possible CoT. When compared with a three link biped walker seen previously, the Atlas robot was found to have a higher CoT. This difference may be due to the disparity in average velocity of the two models, with the three link biped being closer to an average human walking speed, as the P/mg ratio was found to be identical for both. It should be noted however that neither of these models are realistic as factors such as heat loss, friction etc. are not accounted for which would lead to a higher real world CoT. Interesting future work could include a study into parameter tuning optimization for various performance metrics as well as the individual impact of the parameters on robot stability.

# Bibliography

[1] Englsberger, Johannes, Christian Ott, and Alin Albu-Schäffer, *Three-dimensional bipedal walking control based on divergent component of motion*, IEEE Transactions on Robotics 31.2, 2015

[2] Nugteren, G.M., *Process Model Simplification*, Eindhoven University of Technology, 2010

[3] Cost of Transport - Wikipedia, `https://en.wikipedia.org/wiki/Cost_of_transport` (accessed: 15.11.2021)