Spider Robot Trajectory Optimization using Factor Graphs

Disha Das
Georgia Institute of Technology
Atlanta, USA
ddas71@gatech.edu

Tarushree Gandhi
Georgia Institute of Technology
Atlanta, USA
tgandhi9@gatech.edu

I. Introduction

In this project, we executed trajectory optimization of a spider robot model using the GTDynamics library [2], a library that allows implementation of kinodynamic constraints on robot configurations using factor graphs.

II. Structure of Implementation

Our implementations were made in the branch *feature/spider/walking* of GTDynamics library. The spider robot model example is located in *GTDynamics/examples/example_spider_walking* folder. The functions of some important files are given below.

A. *spider.sdf*

This file contains information regarding the structure of the spider, its joint angles, joint limits, link lengths, etc in SDF format.

B. *main.cpp*

This file uses the GTDynamics library to generate a .csv file containing information about the joint angles, velocities, accelerations and torques for each time step in the entire generated walk trajectory. This is implemented in the following steps:

- Phases of the spider walk cycle are created.
- These Phase objects are used to generate a WalkCycle object.
- This WalkCycle object is then used in a Trajectory object which then returns information regarding phase contact points at each phase.
- For each phase in the trajectory, the contact point objectives are given as priors in the factor graph.
- The base link objectives are added in the factor graph.
- Link boundary conditions and joint boundary conditions are added in the factor graph.
- Using Levenberg Marquardt algorithm, the factor graph is optimized.
- The results of the optimization are used to generate the *forward_traj.csv* file.

C. *forward_traj.csv*

The *.csv* file generated by *main.cpp*.

D. *sim.py*

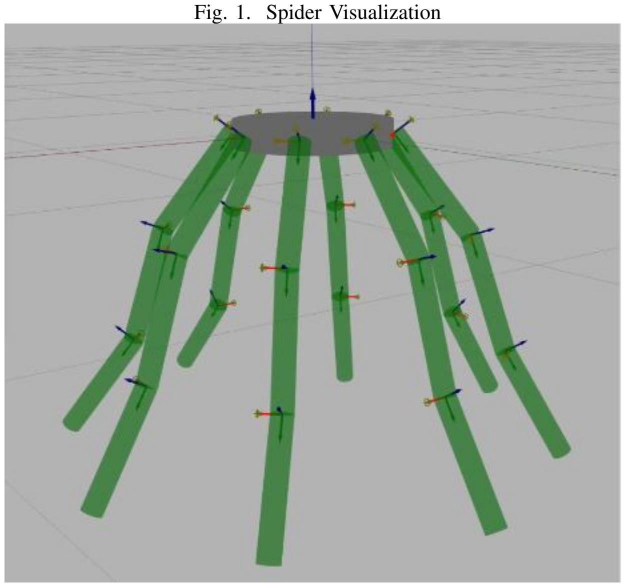
Using Pybullet, we load the environment file, which is in the form of a *.urdf* file, and the spider model, which is the *spider.sdf* file. Then the *forward_traj.csv* is read. At each iteration of the simulation, the joint angles and joint velocities are read from *forward_traj.csv* and these values are used to set the motor controls of the loaded spider model. Thus, using Pybullet, the simulation of the walking motion of the spider is made.

III. Progress Timeline

A. *Spider Robot Visualization in Pybullet*

Previously, we had created a Spider Robot model in SDF format. GTDynamics library was used to generate a traj.csv file for a simple stationary spider. This file would then be used in Pybullet to read joint angle values and joint velocity values at each time step to guide the movement of spider robot model.

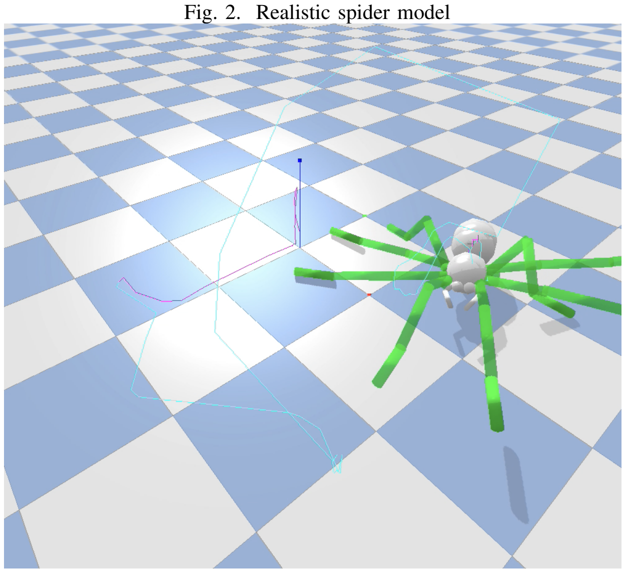
Fig. 1. Spider Visualization



B. *Building a realistic spider model SDF*

The Spider robot model we had previously consisted of a cylindrical base surrounded by 8 evenly spaced limbs. To make the model more realistic, we took the help of [1]. The base of the spider was made into a head, thorax and abdomen regions. The origin of the limbs were placed appropriately along the body of the spider. The lengths of the limb-links were varied to fit the proportions of the sections of a real spider limb.

Fig. 2. Realistic spider model



C. *Walking with Pre-defined footholds*

The first gait movement consisted of moving a limb at a time in sequence. The gait consisted of 9 phases. The first was a stationary phase where the spider is stationary. In the next 8 phases, the spider moves a limb starting from the first to the eighth. At each phase except the stationary phase, the contact point at the limb was given a point goal factor. A positive y-axis value was added to the current contact point position in (x,y,z) coordinates to generate the contact point goal position.

D. *Creating a Tetrapod Walk*

For a realistic spider walk, we created the tetrapod walk. This consists of 2 phases being alternated a few times. The first phase consisted of a motion of even legs of the spider. At this phase, we added contact point goal factors to even legs of the spider. These goal positions were created by adding a positive value along y-axis to the current contact point position. The next phase consists of movement of the odd legs only. This was similarly implemented.

E. *Forward walking and Rotation*

Once the Tetrapod walk was successful, we wanted the spider to walk in a straight line. So we varied the noise priors for point goal factors and base link height factor to have smaller standard deviations. We also added priors to the hip joint angles to give the spider a realistic spider walk. We played with the standard deviation values until we arrived at values that made the spider walk in a straight line.

For rotational movement, the above code was used with a minor difference. To turn left, for example, the legs to the right of the spider were given positive movement along y-axis while the legs to the left of the spider were given a negative y-axis movement. The reverse of this was implemented for a right turn.

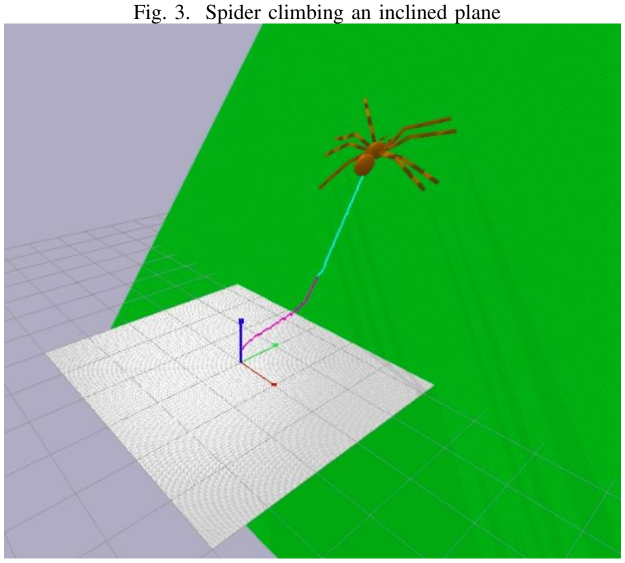
F. *Introducing Constraints at Footholds*

Once the spider was walking in a straight line, we noticed that it’s body was too heavy as compared to its legs to generate enough momentum to propel the entire body forward. A spider walks by making use of tiny hairs at the tip of its feet. These hairs dig into the surface and give the feet a firm grip. Bending the leg at the different angle frees the grip and that’s how, the spider is able to detach its feet from the surface. This mechanism was implemented in Pybullet. When the feet first hit the ground, we introduced contact constraints at the feet that were hitting the ground. This made sure the hitting feet were firmly joined to the ground. When the spider is lifting its feet, the contact constraints in place were deleted. The lifting intention was detected from the traj.csv file, where a positive gradient in the hip angles meant a lifting motion.

G. *Walking on an Inclined plane*

Once the spider was successfully propelling its body forward without allowing the legs to slip, we moved on to the next big step. An inclined place was introduced in the path of the spider. The plane was inclined at an angle of $85°$. The spider was made to walk forward on a horizontal plane and then climb the inclined plane. As it turns out, we were successful in making the spider climb the inclined plane to a height without it slipping. The walk wasn’t always in a straight line motion however.

Fig. 3. Spider climbing an inclined plane



IV. Phase Abstraction

In the process of trajectory optimization, a few classes were created relating to Phase implementations in a trajectory. Given below is a description of each of these classes:

A. Phase:

A Phase class contains information regarding the robot configuration used in a phase, the links that are coming in contact with the ground at this phase and their time duration. For example, in case of Tetrapod walk, some phase objects would be- stationary, even legs stance and odd legs stance.

B. WalkCycle:

A WalkCycle class stores information regarding a sequence of Phase objects. Following the above example, a WalkCycle object called ”tetrapod walk” would consist of the following sequence of Phase objects- stationary, even legs stance, stationary and odd legs stance.

C. Trajectory:

A Trajectory class contains information regarding a Walk-Cycle and the number of times it is repeated. It has helper functions for Trajectory related implementations such as generating contact point objects at each transition, duration of phases, writing the joint values to traj.csv file and many more.

V. Future Work

We were unable to accomplish the task of making the spider walk in a straight line. On top of that, this walking motion with constrained footholds was implemented in Pybullet, and not in GTDynamics. In the future, we would like to implement these constraints within an environment with inclined plane and use GTDynamics optimization on this setup.

References

[1] A. Gasparetto, R. Vidoni, and T. Seidl. Kinematic study of the spider system in a biomimetic perspective. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3077–3082, 2008.

[2] Mandy Xie and Frank Dellaert. A unified method for solving inverse, forward, and hybrid robot dynamics using factor graphs, 2019.