QuadraTot

Code Review 1

Diana Hidalgo, Sarah Nguyen, and Jason Yosinski
{djh283,smn64,jy495}@cornell.edu

November 2, 2010

1 Executive Summary

We are using machine learning to design a gait for a quadruped robot. The robot has an on-board computer, drivers for the motors, and will have camera feedback. The code so far consists of the robot class, optimization class, parameterized model, sine model, and camera feedback. Both the hardware and software are progressing on schedule. Some promising gaits have been generated, and we anticipate a final evaluation of the robots gait in November.

2 Proposal Summary

We are using machine learning to design a gait for a quadraped robot from Hod Lipsons lab. The metric for evaluation of the designed gait is speed.

3 System Architecture Summary

The quadraped robot has an on-board computer running Linux. The lower level drivers are in C and we are implementing the system in Python. Feedback about distance travelled is provided via a Wii remote.

4 Implementation Status

`Robot` class

- *Team member:* Jason.

- *Current implementation:* Class wrapper for commanding motion of the robot. The `Robot` class takes care of the robot initialization, communication with the servos, and timing of the runs. In addition, it prevents the servos from ever being commanded to a point outside their normal range (0 - 1023) as well as beyond points where limbs would collide with parts of the robot body. The main class function, `run`, accepts a motion model (any function that takes a time argument and outputs a 9 dimensional position) and will run the robot using this motion model, including, if desired, smooth interpolation over time for the beginning and end of the run.
- *Future improvements:* None planned, though some will probably arise.
- *Status:* 90% complete.

`Strategy` and `RunManager` class

- *Team member:* Jason and Sarah.
- *Current implementation:* The user has a choice between three different learning strategies: uniform random hill climbing, gaussian random hill climbing, and N-dimensional policy gradient algorithm descent.
  - *Uniform random hill climbing:* Creates a random neighbor by randomly choosing one parameter to adjust, and changing either completely randomly in UniformStrategy(), or slightly in UniformSlightStrategy(). Then evaluates this neighbor by running the robot with the newly chosen parameters. If this neighbor results in a longer distance walked then the previous best neighbor, we save this neighbor as the new best neighbor. We then repeat the whole process using the best neighbor as a base.
  - *Gaussian random hill climbing* Creates a random neighbor byi changing each parameter randomly based on a normal distribution. Then evaluates this neighbor similarly to uniform random hill climbing.
  - *N-dimensional policy gradient descent* Estimates the policy gradient by evaluating t randomly generated policies near an initial parameter vector. Then computes the average score
- *Future improvements:* Machine learning of objective function.
- *Status:* Described algorithms coded. Machine learning of objective function to be coded by Jason.

`Parameterized Motion Model` abstract base class

- *Team member:* Jason.
- *Current implementation:* Several functions exist, but none are within a class framework.
- *Future improvements:* Make into class.
- *Status:* 40% complete.

`SineModel` classes

- *Team member:* Jason.
- *Current implementation:* Commands motors to positions based on a sine wave, creating a periodic pattern. Parameters are: amplitude, wavelength, scale inner vs outer motors, multiply left-right, multiply back-front, shift front-back, shift right-left. Currently a function, not a class.
- *Future improvements:* Make class, add time offset parameters. Continued tweaking of this class and other motion models will be a main goal of the next month.
- *Status:* 30% complete.

Other derived motion model classes

- *Team member:* All.
- *Current implementation:* Several versions of a SineModel exist, each with different parameters, though solely as functions (not as derived classes of `ParameterizedMotionModel`).
- *Future improvements:* Implementing and tweaking current and new motion model classes will be our primary effort over the next month.
- *Status:* 20% complete.

`WiiTrackClient` and `WiiTrackServer` classes and hardware

- *Team member:* Jason and Diana.
- *Current implementation:* A Wii remote tracks the location of the robot through an infrared LED mounted on top of the robot. We used CWiid library which is able to interface with the remote via bluetooth to determine the location of the infrared LED. This information is accessed through some functions we wrote and passed to

the program. The program gets the robot's position at the beginning of each run and then again at the end. The program then calculates the net change in position and uses that as the distance walked.

- *Status:* Complete.

5 Evaluation status

- After running our program and finding a good gait, we will then evaluate our system by the distance walked in a certain amount of time. This evaluation metric depend on the result of running our program for many iterations. We have done 50 runs of 3 different inital parameter vectors for random hill climbing, gaussian random hill climbing, and policy gradient descent. The evaluation for our particular project is an ongoing process, but we anticipate a final evaluation taking place early-to-mid November. The evaluation process will be completed by Jason, Sarah, and Diana.

- We also coded `explore_dimensions` to collect data for dimension varying plots. We intend to do further explore this by running `explore_dimension` around a parameter that produces a medium-good gait.

6 Project schedule timeline

We are meeting our project timeline. We have developed two new algorithms, gaussian hill climbing and policy gradient descent and tested all three of our major algorithms on the robot. We have also quantified our current results. In the next month we hope to develop one more learning strategy, machine learning of an objective function.

7 Schedule

Below is a rough schedule of fixed external deadlines and anticipated team milestones.

	Milestones	Deadlines
Week 1 (9/13-9/19)	~~Read papers, get lab access, talk to relevant other researchers~~	9/17 Final proposals due
Week 2 (9/20-9/26)	~~Continue reading, get robot to move~~	
Week 3 (9/27-10/03)	~~Implement parametrized gait and determine proposed coding schedule for more advanced algorithms in time for Code Review #1.~~	
Week 4 (10/04-10/10)	~~Begin main algorithm dev/testing effort~~	10/5 Code Review #1
Week 5 (10/11-10/17)	~~Algorithm dev/testing~~	
Week 6 (10/18-10/24)	~~Algorithm dev/testing~~	
Week 7 (10/25-10/31)	~~Algorithm dev/testing, quantify/solidify current results for Code Review #2~~	
Week 8 (11/1-11/7)	~~Finish collecting results, begin writing~~	11/2 Code Review #2
Week 9 (11/8-11/14)	Algorithm dev/testing	
Week 10 (11/15-11/21)	Finish collecting results, writing, get final demo ready	
Week 11 (11/22-11/28)	Finish collecting results, writing, get final demo ready	
Week 12 (11/29-11/30)	Final demo	11/30 Final presentation

8 Activity log (Optional)

- *9/13-9/19:* Set up. Set up trac, worked on pre-proposal and presentation, got lab access, met with Jim Torresen, contacted Juan Cristobal Zagal, read papers, got the robot to walk using previously coded program. Questions – How do we move the robot? How do we command the motors?

- *9/20-9/26:* Moving the robot. Installed Linux on the robot, figured out how to command the motors, coded a program to make the robot stand up, coded a simple motion pattern. Questions – For initial program, what parameters and model should be used? How do we limit the search space? What algorithm should we start with? How will the robot get information about how far it moved?

- *9/27-10/3:* Initial program. Chose several parameters and created a simple parameterized sine model, implemented a random hill climbing algorithm to optimize the parameters, ran some trial iterations, began setting up the cameras for feedback. Questions – Did the initial program work? What parameters are important? Are additional parameters necessary? What algorithm should we implement for the final program?

- *10/4-10/10:* Plan for final implementation. Discussed possible algorithms to use and new parameters to add. Worked on report for code review. Discussed feedback from code review. Questions – Advantages for gradient descent? Should we start with a completely random parameter vector or should we start with one that results in motion and let the program optimize it? How does the camera system work?

- *10/11-10/17:* Discussed how we were going to proceed with the project and set up a more detailed schedule. Got the cameras working and calibrated them. Researched the 3 available streaming protocols. Questions – Which streaming protocol would work best? Which improvements should we start with?

- *10/18-10/24:* Created a variation on the random hill climbing algorithm to choose the next neighbor based on a Gaussian distribution. Tried to work with two of the streaming protocols for the camera system, but were unsuccessful. Decided to use a Wii remote for positional information instead. Wrote the integration code for the Wii remote. Refactored our code to separate the different algorithms. Began planning the algorithm for a modified gradient descent. Questions – What values should we use in the gradient descent algorithm? How much should the each parameter be changed? How many neighbors should be sampled before choosing the next state? How can we compare the results of our different algorithms? How repeatable is the distance traveled for a given parameter vector?

- *10/25-11/2*: Created a modified gradient descent algorithm. Ran 150 trials of the three algorithms. Each algorithm started with a specific parameter vector and ran for 50 iterations. Created and ran a program to get a feel for the overall look of the search space. Ran some trials to see how reproducible the distance traveled was. Prepared for code review 2. Questions – What does the search space look like? Are there a lot of spikes, or is it fairly smooth? Based on the results of the algorithms, what conclusions can we draw?

9 Proposal Modifications

We are not deviating from our original proposal.

10 Appendix

A brief description of the code uploaded to the CMS follows:

- `optimize.py`: Determines a strategy to try and runs the robot with that strategy.
- `RunManager.py`: Deals with all the details of running the robot, including choosing an initial parameter, running the robot multiple times, tracking distance walked, and writing to the log file. Also includes the `explore_dimensions` method.
- `Strategy.py`: Contains all the different possible strategies, which will be passed as objects in `optimize.py`.
- `Robot.py`: Implements the `Robot` class, described in Section 4.
- `SineModel.py`: Implements a sine based motion model, described in Section 4.
- `Motion.py`: Motion helper functions.
- `WiiTrackServer.py`: Broadcasts the position of the infrared LED¿
- `WiiTrackClient.py`: Connects to the WiiTrackServer to get the current position information.