Re-Learning to Walk: Adding Force Feedback Control to the Quadruped Robot

Mark Quilling Andy Gooden
In collaboration with
Andrew Ng Honglak Lee Yirong Shen

Introduction:
Currently, the quadruped robot plans a sequence of foot movements and joint angles using a two level hierarchical decomposition of the task. The high level planner plans a sequence of "target foot placements" to achieve a desired goal location from a given start location. The high level controller is implemented using a value function approximation and beam search. The low level controller plans a series of joint angles to execute the series of footsteps planned by the high level controller. It is the job of the low level controller to keep the robot balanced, prevent obstacle collision, and execute the desired footstep. The low level controller is implemented using a policy search whose policy parameters were learned. Currently, there is no feedback in the control path, and a successful journey of the quadruped is highly dependant on the accuracy of the simulation model. Our project will add force sensors to each of the quadrupeds feet, close the control loop, learn new parameters for the low level controller's optimal policy, and implement on-line trajectory planning.

Current control system:
The low level controller uses a potential field to determine the optimal trajectory to accomplish each individual footstep. The controller uses the following potential field desription:

$$U_t(\Omega_t, u_t^i, u_g) = U_t^g(u_t^i, u_g) + U_t^s(u_t^i, u_g) + U_t^c(\Omega_t) + U_t^o(\Omega_t)$$

Where $U_t^g(\cdot)$, $U_t^s(\cdot)$, $U_t^c(\cdot)$, and $U_t^o(\cdot)$ represent potentials for the goal, floor, center of gravity, and body orientation, respectively. At each instant of time, the robot computes the negative gradient of the potential function evaluated at the current state.

$$\tilde{g}_{t,j} = -\nabla_{w_j} U_t(\Omega_t, u_t^i, u_g)$$

The robot then tries to execute the change in joint angles given by $\tilde{g}_{t,j}$. This, however, does not guarantee that the three stationary feet do not move. To prevent this, $\tilde{g}_{t,j}$ is projected into the null space of Φ_t, which is a matrix whose nine columns are the gradients of the state variables with respect to the x,y,z coordinates of each of the three feet. This null space projection prevents the x,y,z coordinates of the stationary feet from changing, while still allowing the desired trajectory to make progress down the negative gradient of the potential field. This process is all executed off line and the joint angles are written to a file to be later played back, open loop, to the robot. There are many problems that can arise from assuming the environment is static and that we can approximate it exactly. The new control will incorporate force feedback to remedy the problem of uncertain terrain. New control system:

The new closed loop control takes in to account the feedback obtained from the force sensors. By using this information we are able to better navigate uncertain terrain correcting for unknown parameters of the environment. By introducing real world information, through a potential function, we are able to increase the confidence the robot has about the current state. Only a few things where modified from the above algorithm:

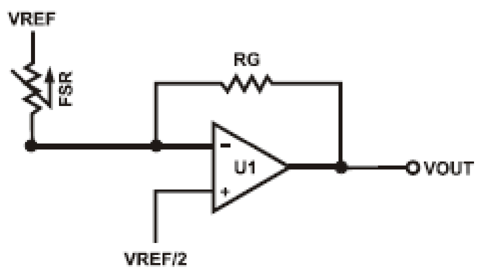
1. We now write the control action directly to the robot instead of a file.
2. Update current state reads in the current force reading from the sensor via a micro controller.
3. We added a potential function proportional to the force.

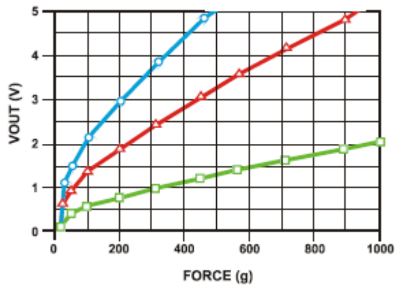
With these three added capabilities we are able to do all the control decisions online in real time and have access to data from the real world.

Serial Communication:
The previous system used serial control code on a PC for one-way communication with the robot. The code would initialize the port, initialize the robot, read the pre-computed joints.txt file, and control the one-way flow of servo commands to the robot.

In the new system, we ported the robot initialization code over to Linux and integrated it with the low-level controller. Additionally, we added new two-way communication code so that the force sensor data could be read and sent to the force potential functions. Finally, we implemented real-time control, such that the next optimal joint trajectory was computed after receiving the packaged force sensor data from the robot via serial.

Microcontroller and force sensor circuit
To drive the force sensing resistors (FSR's), we created a bank of opAmps. The figure below shows our circuit. The circuit converted the FSR readings to a voltage and sent the signals to the microcontroller.





The microcontroller is a 28 pin basic atom pro with A/D capabilities. The purpose it serves is to continuously monitor the sensors, package the data and ship it up to the computer. It uses a basic serial communication protocol to communicate with the linux box. Each sensors reading is packaged into 2 bytes before sending.

Force feedback using estimated desired force:
The approach we took combined the calculation of the desired force at the end of a task with the estimate of the force gradient with respect to each state. By using the foot placement and center of gravity we are able to predict a force that the robot should feel on each foot. For every task we calculate a desired goal force which we use to create an attractive potential around, call this Fd_goal. For every iteration of the algorithm (each task contains many) we also calculate the desired force on the robots feet at that instance of time based on the current state, call this Fd. The next step of the algorithm is to calculate the gradient, to do this we perturb the current state and re-calculate the desired force on each foot, call this Fd_prime. We then estimate the gradient, δF, by the difference between Fd and Fd_prime. Our potential function for the force is then constructed as follows:

$$c1 * \left\| Fd\_goal - (Fa + c2 * \delta F) \right\|^2$$

Here c1 and c2 are the learned parameters from the PEGASUS search algorithm and Fa is the actual force reading from the sensors.

Fd_goal:
The calculation of this force is done once each time we start moving a new foot. The force balance equations we solve are as follows:

$$\begin{bmatrix} Wcg_x \\ Wcg_y \\ W \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} x_3 f_3 \\ y_3 f_3 \\ f_3 \end{bmatrix}$$

Here W is the mass of the robot, cg is the center of gravity Cartesian co-ordinates, (x_i, y_i) i = {0 1 2} are the x and y co-ordinates of the stationary feet, f_i i = {0 1 2} are the forces on the stationary feet, (x_3, y_3) and f_3 are the co-ordinates of the moving foot and force of the moving foot after task completion. As one may have noticed the force balance equations have 4 unknowns with 3 equations, thus the system is over determined and has infinite solutions. To remedy this problem we assign a force to the moving foot, F_CONTACT, which is a small contact force that the robot should feel on the moving foot before continuing to the next task. The above linear equations can then be solved using a simple matrix inverse. The solutions to this equation, including the assigned moving foot force, are Fd_goal. A few downfalls to this calculation is that we don't have access to the cg location at the end of a task so we assume it remains about constant throughout the maneuver, this is clearly wrong but seems to work ok. Our calculations using the above equations are pretty accurate and provide a concrete goal for the robot to aim for.

δF:
This quantity is what actively changes throughout the gradient calculation. It is composed of two components, Fd the instantaneous calculated force of the current state (if our calculations and sensors where perfect then this would be Fa), and Fd_prime the perturbed force. Lets first discuss Fd, the reason we use our calculated force instead of the sensor readings is due to the following facts: 1. the sensors we are using are very inaccurate, 2. when we try to estimate the gradient we want to be able to base our calculations in the same reference frame i.e. use the same equations for both perturbed and un-perturbed state. So for every gradient calculation we calculate Fd based on the current state of the robot and use this quantity for every perturbed state. Fd is calculated in the same way as Fd_goal except here, because the foot is moving, we set f_3 to be zero. Now we perturb the current state one component at time calculating the value of the potential field at that new state. When one component of the state is perturbed we recalculate the total state and again perform a force calculation. The difference between this force and the Fd is δf. We can think of δf as the amount of force we expect to get if we move a certain component of the state be a small increment. This quanity is what tells the robot the direction in which to move to achieve the desired force.

c1 and c2:
These are the learned parameters from the search. The constant c1 is the relative weight with respect to the other terms in the potential function, it basically says how important force is compared to our other objectives. The constant c2 is the force gain. We use this to amplify the direction of δf because in many cases δf may be very small and almost have no effect on the potential. By amplifying the change we are able to induce a larger variation in the potential over the different perturbations.

A nice way to think of our potential function is that we have some desired force at the bottom of some quadratic "bowl", Fa represents a point along one of the sides of the bowl, and δf is a direction from Fa. So by trying to minimize the above quadratic function we are trying to find δf which points towards the goal. If our only task was to maintain a given force distribution over the legs (see results section) then this potential would be ideal, but because the robots state is dynamically changing over time we have other goals that become more important. When a given leg is trying to reach a goal configuration there are other factors that contribute to the potential function, this is where learning the weights will determine the best relative weighting between the different potentials.

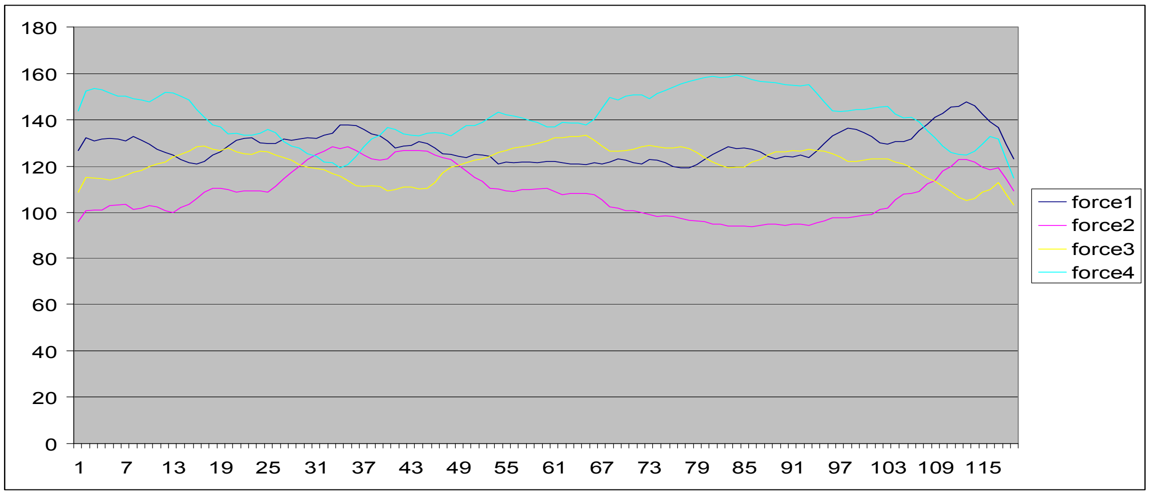
Problems:
We encountered quite a few problems with the implementation of the force potential. Due to very noisy sensors and inaccurate state description we had a hard time getting the force to converge to the desired.

Sensors:

The sensors we used where very insensitive to low forces. Due to this fact we almost had a binary model of the contact force and the above potential was almost overkill for this model. On top of the insensitivity these force sensors gave very sporadic measurements where the force would jump from high to low values and visa versa. In theory with accurate, low noise, force readings the above potential will work very well, but in our experiments we had a lot of problems getting a good response from the robot because small movements would cause large force changes and thus create a large potential.

Force calculations:
In all the force calculations we assume a stable robot as well as the state that we think we are in is the actual real world state. Because there is really no feedback from the real world (besides the force) we often have a very inaccurate representation of the world. Because we rely on this representation to calculate the expected forces on the feet we often get very inaccurate force predictions. With a perfect model of the world, our instantaneous force predictions should equal the measured force. Although these calculations were inaccurate we were still able to achieve decent performance from the robot.

Results:



Force measurements: trying to maintain balance on flat surface

The above plots are of the robot trying to maintain a equal force distribution over all feet in different situations. The first plot the robot is on a tilted surface and tries to reach and equilibrium state. Because of the inaccuracy of the sensors the robot often never finds this state due to “of on” nature of the sensors. Through observation we see the robot reacting to different situations and trying to adapt by changing its position. The next figure the robot is on a flat ground and is simply trying to maintain equal force on each foot. The results are a little better but the sensors still make it quite difficult to get a clear picture of the force converging to the goal. When we tried to collect data of the walking robot we had a lot of trouble collecting reliable, representative data sets, they were basically noise. We where able to successfully navigate a multi level surface using force feedback (see video) and gain insight that our algorithm was working correctly. One thing is clear from all the test and trial we conducted: we need better sensors! Given the time constraints of the quarter by the time we realized the sensors where not adequate for the robot it was too late.

Conclusion:

In conclusion our project turned out quite well and we were able to demonstrate its performance on real hardware. The algorithm used was sufficient to produce the desired response but there are quite a few more improvements that need to be made. First is we need to get better, more accurate sensors on the feet. This follows directly from the fact that our potential function relies strongly on accurate force measurements that are smooth and continuous. As of now a simple algorithm, such as moving the target down in the z direction until the moving foot touches would provide almost the same results. What we where interested in was providing a dynamic response which would not only adapt to uncertain terrains but stabilize the robot throughout its task. One thing that our potential function relies heavily on is the accuracy of both force calculations and measurements. The first we have a pretty accurate set of equations for force under normal circumstances, but if the robot is in an unstable state or has not centered correctly then we may assign negative numbers to the force calculations. Although these circumstances are rare, once they happen the robot is in big trouble. One of the most frequent times this happened was when we are transitioning from centering to moving and the cg is just on the border of the supporting triangle. Because we assign a zero desired force to the moving foot, the robot in-turn thinks it need to apply a negative force to a leg to stabilize itself…this is clearly wrong. Another time we may calculate negative force is when we are assign the goal force to each leg. Because, at the moment, we don't have access to the goal cg we assume the cg remains about constant throughout the maneuver. This can easily be remedied by having the high level planner spit out the desired cg as well as the feet positions. The algorithm should be modified to recognize situations where it may see negative force calculations and adapt accordingly. In closing the we successfully designed a closed loop force feedback controller for the quadruped robot and successfully demonstrated it performance.

Appendix: Cool Pictures