Creation of a Fallback Catch Method

Megan Berry

Mechanical Engineering Senior

MAE 490 (4 credits)

Abstract

In order that the Cornell Ranger remains autonomous during long distance record attempts and avoids damage to the system it is desirable to add a method to the programming of the robot which protects it from falling backwards. It was required that the method always catch the robot given indoor track conditions and desired that with future versions the method would also reattempt push off. The fallback catch method works by monitoring the angular rate of the stance leg which should always be positive for regular walking mode. When this value goes negative the swing leg moves to a given angle behind the stance leg and uses the resistive torque of the swing leg foot motor to dampen the fall impact of the robot.

Control of Failure Modes

For the Cornell Ranger, a failure of the system is defined as a failure to walk and occurs when the robot falls over or if the robot stands still. The three legged biped design of the robot eliminates the possibility of falling to the left or right by providing side stability. Therefore, the main modes of failure are in the robot falling either forwards or backwards. Because the purpose of the Cornell Ranger is to set long distance records, or to be highly reliable, it is important that programming methods be created to limit these falling failure modes. Falling forward is actually a desired action because walking is essentially the act of falling forward and catching yourself repeatedly. However, this falling must be controlled and the robot must be able to catch itself by taking an appropriately sized step and planting its foot to prevent a complete fall. Falling backwards is always an undesired action and until now there was no measure to prevent a fall backwards other than pulling up on the safety cable and thus breaking the autonomy of the robot.

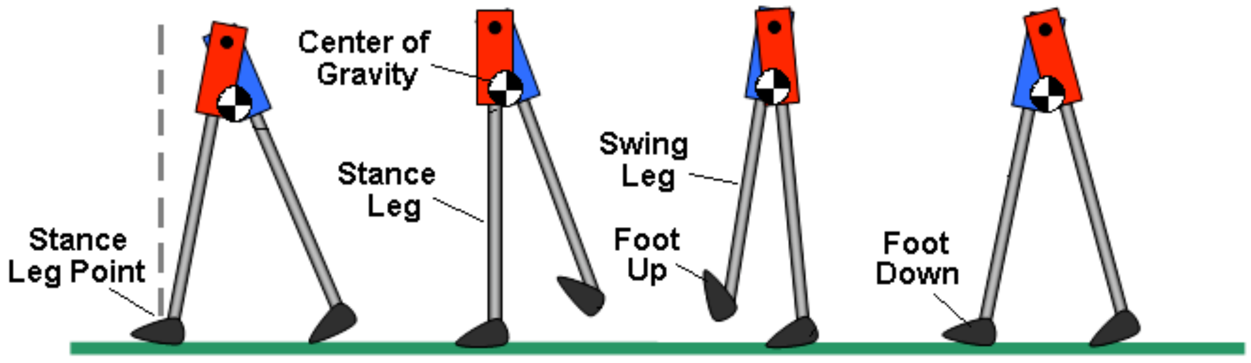


Figure 1: Standard Walking Mode and Definitions

Falling Forward: Causes and Preventative Measures

The initial control system for the Cornell Ranger had a static step size that was adequate for level, undisturbed walking. However, inconsistencies and even slight declining slopes in the walking surface would cause the robot to fall forward. The center of gravity in the hip of the robot would move forward faster than the feet were able to travel for a static step size. Without the ability to adjust the step size the robot would quickly become critically unbalanced and fall forward. In order to prevent the forward falling behavior a gyro system was added to the robot to recognize when the robot center of gravity speed was exceeding what could be supported by the current step size. When the robot receives input that the hip speed is outside a given range it adjusts the step size accordingly. If the speed is too high the target step size is increased so that the swing foot always contacts the ground forward of the robot center of gravity. If the downward slope of the walking surface is too great or the robot is impacted by a large force, either a pushing force from behind or a pulling force from the front, then the robot my not be able to catch itself due to mechanical or imposed limitations of the system. One mechanical limitation is the speed at which the motors can swing the foot and leg forward in order to produce a larger step size. This speed is limited by the motor design and the power available to the motor from the robot batteries. A secondary mechanical limitation is the wire which connects the inner and outer set of legs and prevents the angle between the two from being too great. The programming of the robot also limits the step size to what the robot is able to push off from in order to accomplish the following step. If the legs are split too far apart the back set of legs will not be able to put enough energy into the system, through the push off, in order to move the center of gravity forward of the stance legs. Because the robot would still try and swing the leg forward after push off, this would cause the robot to fall backwards. However, this is only one example of a situation that could lead to the falling backwards failure mode.

Falling Backward: Causes and Goals for Prevention

Now I will discuss the most likely causes of failure by falling backwards. As explained above, the robot could fall backward after taking such a large step that the energy added in push off is insufficient to propel the center of gravity forward of the stance leg contact point. Falling backwards could also occur even for normal step size if the push off force is set to a very low value. This is typically not a danger during normal walking, but this failure method was induced by purposely setting the push off torque to a low value in order to test the effectiveness of the falling backward catch mechanism. The most probable cause of falling backward would be if the upward slope of the walking surface is too great or if the robot is impacted by a large force, either a pushing force from the front or a pulling force from behind. Figure 2 illustrates how a regular walking mode can become a backwards falling mode after a disruptive force is applied to the robot. The yellow arrows show the directions of angular momentum for the two legs and their rough magnitudes at different points in the walking cycle. Because of the gyro system the robot will respond to these disturbances by decreasing the step size, however there is also a minimum step size limitation that is given in the programming of the robot. If a decrease in step size is not enough to correct for the disturbance then the center of gravity will move behind the stance point of the legs that are farthest back. If this happens and the robot center of gravity does not have sufficient momentum to swing forward of the stance point then the robot will fall backwards. Unlike falling forward, where step size can be adjusted in order to compensate for a disturbance which causes the robot to fall forward, falling backward was previously an uncorrectable failure mode once it had started to occur.

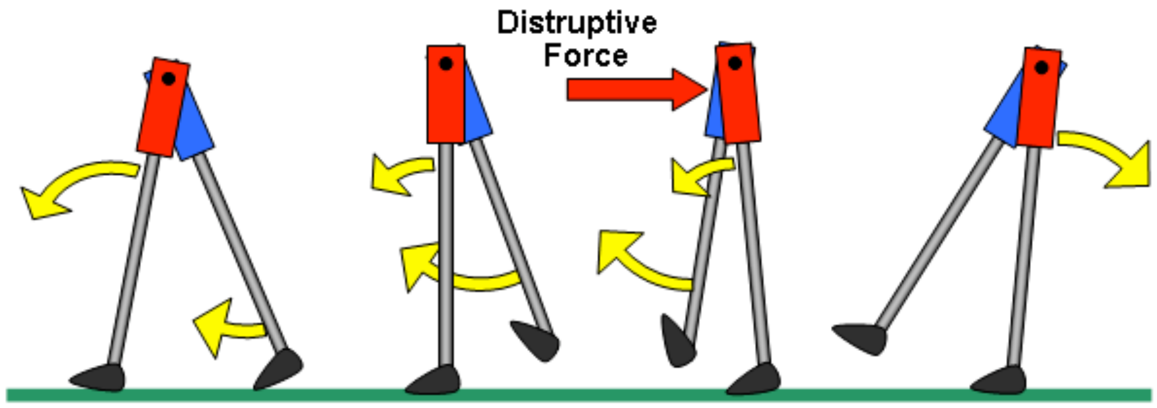


Figure 2: Angular Momentum Directions and Magnitudes in a Backwards Fall

For the purposes of setting a record and to display our confidence in the robot's autonomy it is desirable to not be touching, or even holding a slack wire, while walking the robot. In order to do this, and feel secure that the robot would not potentially be damaged, the robot required a method to be added to the robot programming that would provide protection against the robot falling backwards. The clearest way of doing this was to program the robot to place its swing leg back behind the stance leg if the stance leg begins to swing backwards. This is effectively like telling the robot to take a step backwards, which is generally what a person does to balance themselves when they experience a strong push backwards. The leg then catches the robot and prevents it from continuing to fall backwards. This fallback catch method was designed with the assumption of indoor track walking conditions. This insinuates very small slopes in the walking surface, few surface disturbances, and no large forces exerted on the robot body. With these assumptions, the method was created with the initial goal of catching and stopping the robot from falling backwards in order to protect the robot from damage. Other cases, such as a significant applied force from a push or pull on the robot body or attempting to retry push off after catching itself from a fall, are considered in the section of future work.

Design of the Fallback Catch Method

The fallback catch method needs to be activated whenever the robot begins to fall but must never activate unnecessarily. The data coming from the two gyros gives the angular rate of rotation for each leg. A positive angular rate corresponds to the top of the leg swinging forward, which is always the case for the stance leg during regular walking, and a negative angular rate corresponds to the bottom of the leg swinging forward, which is always the case for the swing leg during regular walking. The data called AngleRateStanceLeg always gives the angular rate for the leg, inner or outer, that is currently the stance leg. If the AngularRateStanceLeg goes to a negative value it is certain that the robot is no longer in the regular walking state and has begun to fall. However, as a precaution I added a timer which requires that the AngleRateStanceLeg read a negative value for 50 cycles in a row, or for 50 milliseconds. This protects the robot from going into falling mode if there are momentary spikes or errors in the data. Additionally the robot should not go into falling mode while it is being positioned to begin walking. For this reason, a second condition of the step number not being equal to one was added. This allows the user to rotate and move the robot as much as necessary as long as the initial swing leg has not yet experienced impact. The third condition is that the robot is not already in falling mode, this keeps the robot in falling mode even though the AngleRateStanceLeg will stop reading a negative value once the robot catches itself from falling. If all of these three conditions are met the hip state and foot state are set to falling mode. If one of these three conditions is not met the robot will check to see if the robot is already in falling mode.

Once in falling mode the robot effectively enters a different state where it ignores all the main walk controller code and awaits the occurrence of different triggers in the falling backward catch method. The first action is triggered when the robot enters the falling mode. The foot torque for the swing leg is set to pull up the foot and hold it in the upwards position. This allows the foot to swing backwards without scuffing. Second, the hip torque for the swing leg is set to swing the leg back behind the stance leg to a target fallback angle. While the swing leg is being moved the program does nothing but wait for the angle between the two legs to come within a given angle to the target fallback angle. When this occurs the foot torque for the swing leg is set so that the foot swings down and holds in position at a neutral angle. The robot then lands on the extended foot and breaks its fall. See Figure 3 for an illustration of this fallback catch method and Figure 5 for a flow chart diagram of the method. Note that green arrows in the flow chart correspond to actions taken when the if conditions are true and red arrows to actions taken when the if conditions are false. Although this method was very effective in catching the robot from falling backwards it was noted that the robot generally lands quite hard on the swing leg and causes both the front and back legs to bounce off the ground several times. In order to soften the impact on the robot and recover from the fall more quickly, I added another section of code that uses the swing leg foot like a damper as the leg takes the impact of the fall.

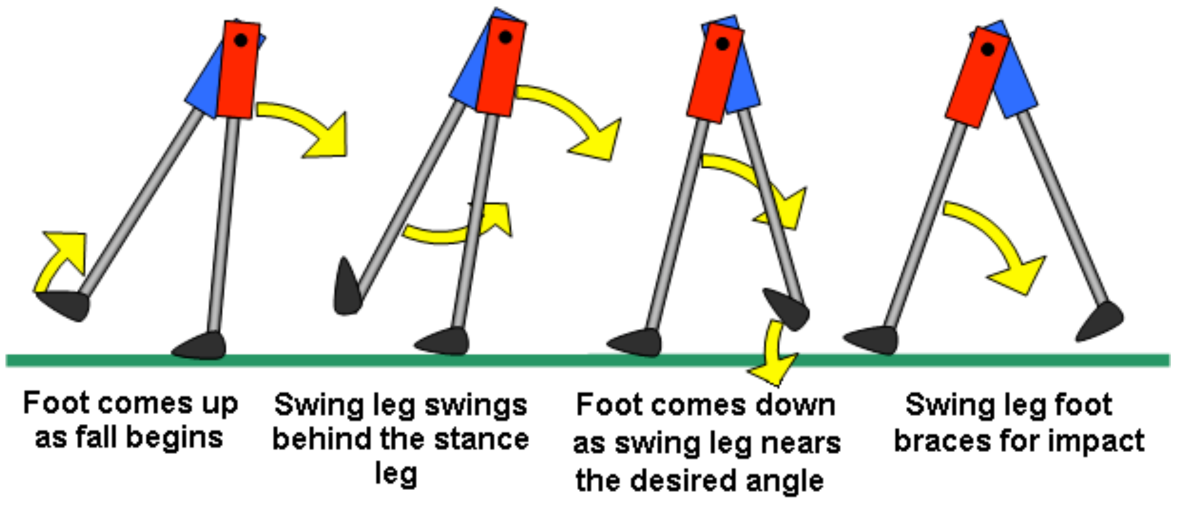

Foot comes up as fall begins
Swing leg swings behind the stance leg
Foot comes down as swing leg nears the desired angle
Swing leg foot braces for impact

Figure 3: Illustration of a Fallback Catch

Addition of a Fall Impact Damper

In this second version of the code an extra boolean variable was added called RearImpact. RearImpact is set to zero when the robot enters the fallback state and set to one when the swing leg feet register an impact. The RearImpact variable triggers a change in the foot torque setting from being used as a damper, to absorb the impact of the robot's fall, to being used as a stiff support for the robot. This version functions in basically the same way as the previous version except that an extra section is added to the fallback catch method. See Figure 4 for an illustration of the different actions created with this method and Figure 6 for a flow chart diagram of this method. Note that the same arrow color convention applies. When the swing leg nears the target angle the foot torque is set so that the foot points sharply downwards. The spring and damper constants for the foot are set to special parameter values that can be changed until the back foot absorbs the energy smoothly and the front foot does not lift off the ground. As the impact of the fall pushes the foot into a neutral position the impact sensor is triggered which changes the RearImpact variable to one. As described previously, the torque setting for the back foot then holds the foot in the neutral position. In this way, the robot is stopped from falling backwards without any unnecessary oscillations in the system. See the Appendix for a full version of the fallback catch method code as it currently exists in the Ranger controller.

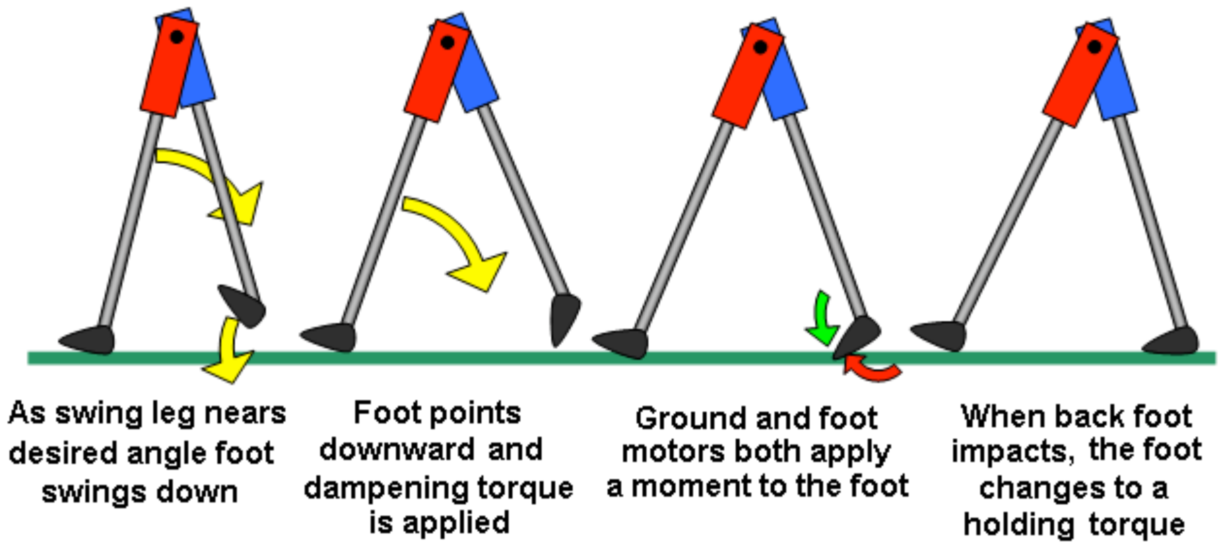

As swing leg nears desired angle foot swings down
Foot points downward and dampening torque is applied
Ground and foot motors both apply a moment to the foot
When back foot impacts, the foot changes to a holding torque

Figure 4: Illustration of a Fallback Catch with Impact Damping

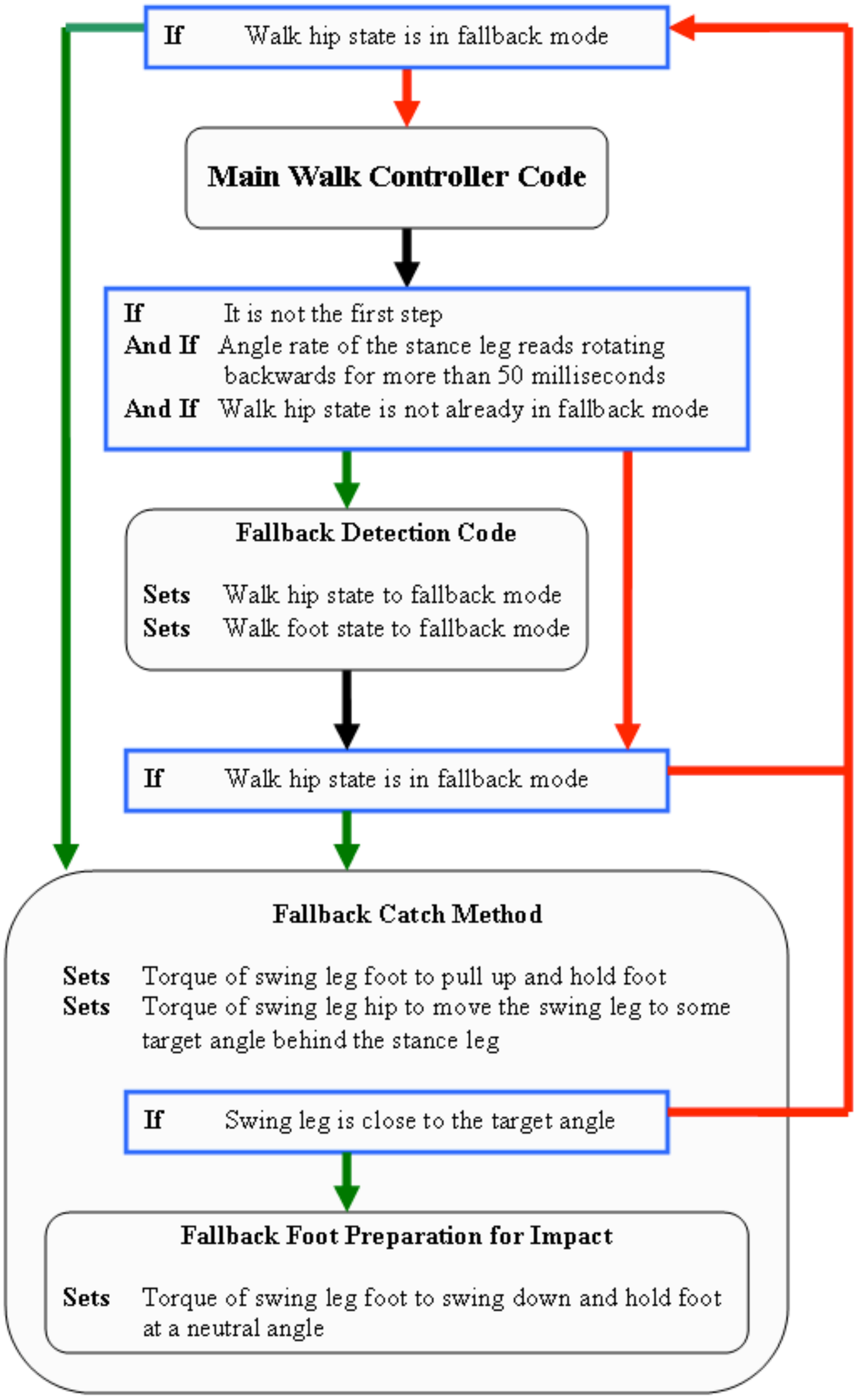
Future Work

Several additional features were tried for the fallback catch method and I feel confident that with another week of work could be made to work. The most desired addition to the system would be a method that attempts to restart the walking mode by giving a maximum power push off with the back feet. Attempts to do this showed that the robot does not have sufficient power to push off successfully from a standstill for the larger hip angles that result when the robot swings its leg backwards in order to catch itself. The robot pushes off and swings forward, but does not have enough energy to get the center of gravity forward of the stance leg contact point. The robot then begins to fall backwards again and reenters the fallback mode. This difficulty could be overcome either by increasing the power to the push off motors or by creating a method that would incrementally pull the legs closer while rocking the robot back and forth. The robot already seems to go into a rock back and forth mode when the code is set to reattempt push off. This action occurred because the robot was passing rapidly between push off and fallback modes. If this method could be made to work it would allow the robot to recover from a fall and continue walking without breaking the autonomy of the robot, therefore allowing it to fall, at least partially, even during a record attempt.

Another feature that I considered would be for a response to a very large disturbance such as a hard push. This method would go in the fallback catch method after the impact of the back feet. It would check to see if the front leg came off the ground for more than a given number of cycles. With the addition of the fall impact damper, front foot liftoff should be reduced to only a second at most. If the front foot was out of contact with the ground for longer than this it could be assumed that the robot was still falling backwards. The back leg would then become the new stance leg and the front leg would become the new swing leg, and the process of taking a step backwards would repeat as necessary. The effect would be that for a large disturbance the robot could take several steps backwards until the front foot remained firmly planted. Additionally, if this system was added the fallback target angle could be reduced. The robot could take several smaller steps backward rather than one large step. This would probably also serve as a solution to the push off problem. The robot could more easily push off from standstill if the hip angle was relatively small.

Other Work

This report reflects work that was begun after the after the Relay for Life. At the beginning of the semester my time was focused on making sure each member of the team had a clear project to work on and assisting the other team members as they began their projects. I worked closely with Yingyi on the installation and integration of the potentiometer system to the existing steering system. I also worked on the revision when the first system failed. The rest of my time was spent testing the robot either in the lab or at Barton, especially during the pushes before the grant proposal decision. For BOOM and Relay for Life the preparation of the robot and posters, organization of the other team members, and attendance at the events took up a week each of my research time. I easily put in an average of 12 hours a week and was present every Wednesday afternoon as an aid to the other members of the team if needed.



If Walk hip state is in fallback mode
Main Walk Controller Code
If It is not the first step
And If Angle rate of the stance leg reads rotating backwards for more than 50 milliseconds
And If Walk hip state is not already in fallback mode
Fallback Detection Code
Sets Walk hip state to fallback mode
Sets Walk foot state to fallback mode
If Walk hip state is in fallback mode
Fallback Catch Method
Sets Torque of swing leg foot to pull up and hold foot
Sets Torque of swing leg hip to move the swing leg to some target angle behind the stance leg
If Swing leg is close to the target angle
Fallback Foot Preparation for Impact
Sets Torque of swing leg foot to swing down and hold foot at a neutral angle

Figure 5: Flow Chart of Initial Fallback Catch Method Code

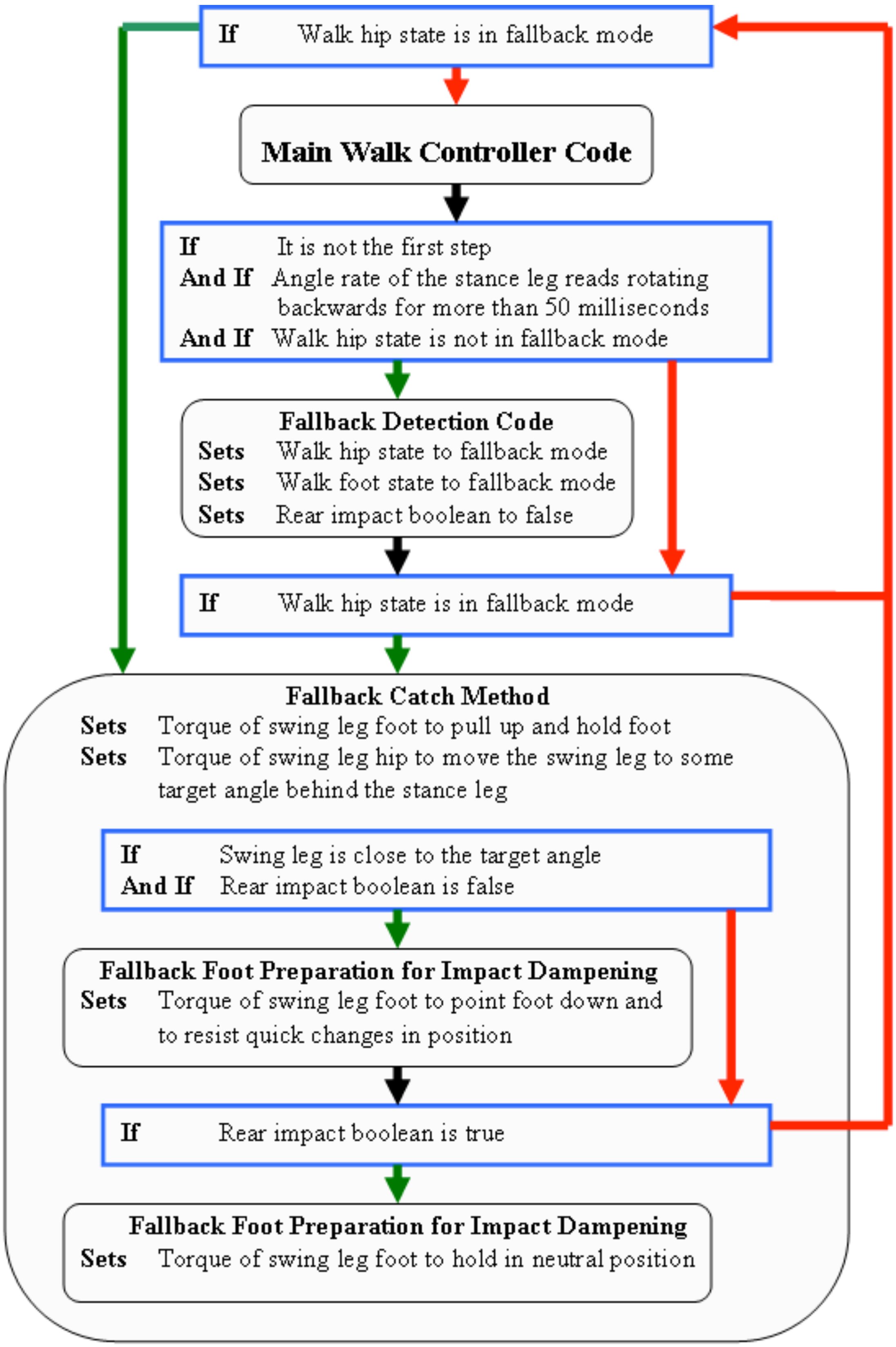

If Walk hip state is in fallback mode
Main Walk Controller Code
If It is not the first step
And If Angle rate of the stance leg reads rotating backwards for more than 50 milliseconds
And If Walk hip state is not in fallback mode
Fallback Detection Code
Sets Walk hip state to fallback mode
Sets Walk foot state to fallback mode
Sets Rear impact boolean to false
If Walk hip state is in fallback mode
Fallback Catch Method
Sets Torque of swing leg foot to pull up and hold foot
Sets Torque of swing leg hip to move the swing leg to some target angle behind the stance leg
If Swing leg is close to the target angle
And If Rear impact boolean is false
Fallback Foot Preparation for Impact Dampening
Sets Torque of swing leg foot to point foot down and to resist quick changes in position
If Rear impact boolean is true
Fallback Foot Preparation for Impact Dampening
Sets Torque of swing leg foot to hold in neutral position

Figure 6: Flow Chart of Second Version of Fallback Catch Method Code