Andrew Nassau
ann8@cornell.edu
MAE '11

Turning Code for the Cornell Ranger Robot

May 22nd, 2010

MAE 4920 Final Report
2 credits Spring 2010

Biorobotics and Locomotion Lab
306 Kimball Hall
Cornell University

800 University Ave
Ithaca NY, 14850
(610)-888-4155

Abstract

The Cornell Ranger's ability to walk long distances relies on its ability to turn. Over the summer and the fall terms, a new turning mechanism had been designed and created for Ranger. It mechanically worked, but it needed to be wired into the Ranger's system. The turning system has to run without significantly interfering with the Ranger's turning as well as being governed by a handheld Remote Control device. A code was generated to run on Ranger's Main Brain and to take the input RC value and use it to set the desired amount of turning. The code has successfully been implemented on the Ranger robot while it is walking.

Introduction

With the new turning mechanism installed on Ranger, the next step was to see if the new steering would actually work while the Ranger walked. At the start of the semester, Ranger was finally ready, mechanically, to walk. As the semester progressed, Ranger came back to life, and as its walk became more reliable, I was responsible for making sure that the steering would work and for generating a code for it.

Code Structure

The basic design of the steering is that the Ranger will twist its inner legs as it is standing on them and rotate the direction of its body. To give the Ranger the largest range of motion, the inner legs would pre-twist one way as the Ranger stepped forward on to them, and twist the opposite direction as the inner feet were on the ground, giving it twice the turning range. The major concerns in writing the code were that the feet had to reach the desired angle in the time it took for Ranger to take a step, and that there would be no twisting during the inner feet's heel strike and push off.

The degree to which the feet would turn will be controlled by a person using a remote control handheld. The steering control allows a human user to guide Ranger around the track without actually controlling the Ranger's walk, only its inner feet. Besides the RC input signal, the steering code would also rely on Hip Angle and Heel Strike to make sure that the feet twisting happened at the appropriate times.

The steering mechanism is controlled by a single motor. The motor has its own microprocessor board, and is controlled by the parameters 'Command Current', 'Stiffness', and 'Dampness'. The current through the motor was governed by these parameters is the following fashion,

□□□□□ □□□□□□□=□□□□□□□ □□□□□□□–□□□□□□□□□x□–□□□□□□□□x lutions to the traction loss should possibly be considered, though the traction will probably be better on the track.

Appendix

Code: steering_fsm.cpp

```
//#include <iostream>
#include <mb_includes.h>
#include "fsm.h"
#include "steering_fsm.h"
#include "steering_fsm_conditions.h"
#include "global_params.h"
#include "global_sensors.h"
#include "global_communications.h"

static float desired_angle;
//static float scaled_rc_value;
//using namespace std;

////////////////////////////////////////////////////////////////////////////////////////////
// Define action functions

int ACT_S_innerlegfront_entry()
{
  desired_angle = -desired_angle; //NEW
```

```
  //set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
  //set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));
  return 1;
}

int ACT_S_innerlegfront()
{
  set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
  set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));

  //set_io_float(ID_MCSI_COMMAND_CURRENT, desired_angle * get_io_float(ID_MCSI_STIFFNESS));

  return 1;
}

int ACT_S_innerlegstance_entry()
{
  //desired_angle = get_io_float(ID_P_S_TEMP_S_TANG); //OLD //Set desired angle

  set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
  set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));

  return 1;
}

int ACT_S_innerlegstance()
{

  //set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
  //set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));



  set_io_float(ID_MCSI_COMMAND_CURRENT, desired_angle * get_io_float(ID_MCSI_STIFFNESS));
  return 1;
}

int ACT_S_innerlegback_entry()
{
// NEW
/*  float alpha = get_io_float(ID_P_S_ILST_S_MAXANG);
  if (get_io_float(ID_P_S_ILST_S_MAXANG)>0.5)
  {
      //give out an error
      alpha = 0.5;
  }
  else if (get_io_float(ID_P_S_ILST_S_MAXANG)< -0.5)
  {
        //give out an error
      alpha = -0.5;
  }*/

  //float alpha = 0; //needs to be removed eventually, 2010/5/22
  //get_io_float(ID_P_S_ILST_S_MAXANG)
  desired_angle = get_io_float(ID_P_S_ILST_S_MAXANG)*(get_io_float(ID_UI_RC_0)/25000 - 3.4);
  //set_io_float(ID_D_S_NULL_S_DANG,desired_angle); Needs to be a data channel in can_id.h 2010/5/22

  set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
  set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));
  return 1;
}
```

```
int ACT_S_innerlegback()
{
   //set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
   //set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));

  //set_io_float(ID_MCSI_COMMAND_CURRENT, desired_angle * get_io_float(ID_MCSI_STIFFNESS));
  return 1;
}

int ACT_S_innerlegswing_entry()
{
  //set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
  //set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));

  //desired_angle = -get_io_float(ID_P_S_TEMP_S_TANG); //OLD
  //ID_D_S_NULL_S_DANG = set_io_float(ID_D_S_NULL_S_DANG,desired_angle);

  return 1;
}

int ACT_S_innerlegswing()
{

  set_io_float(ID_MCSI_STIFFNESS, get_io_float(ID_C_S_NULL_S_ANG));
  set_io_float(ID_MCSI_DAMPNESS, get_io_float(ID_C_S_NULL_S_RATE));

  //desired_angle = -get_io_float(ID_P_S_TEMP_S_TANG); //OLD
  set_io_float(ID_MCSI_COMMAND_CURRENT, desired_angle * get_io_float(ID_MCSI_STIFFNESS));
  //set_io_float(ID_MCSI_COMMAND_CURRENT, 0.1);

  return 1;
}


int ACT_S_stop()
{
  set_io_float(ID_MCSI_COMMAND_CURRENT, 0.0);
  set_io_float(ID_MCSI_STIFFNESS, 0.0);
  set_io_float(ID_MCSI_DAMPNESS, 0.0);
  return 1;
}

////////////////////////////////////////////////////////////////////////////////////////////////


void def_steering_fsm(fsm* fi_fsm)
{

//cout << "defining ankle fsm.. " << endl;

// numStates is the number of states defined in structure.h
int numStates = 5;
// numInpts is the number of inputs defined in structure.h
int numInputs = 8;

// construct the fsm
fsm fsm1(numStates, STATE_S_innerlegfront, numInputs);

char* name = (char*) "steering_fsm";
fsm1.set_name(name);
```

```
// associate the states with actions
fsm1.state_def(STATE_S_innerlegfront, ACT_S_innerlegfront_entry, ACT_S_innerlegfront, ACT_S_innerlegfront);
fsm1.state_def(STATE_S_innerlegstance, ACT_S_innerlegstance_entry, ACT_S_innerlegstance, ACT_S_innerlegstance);
fsm1.state_def(STATE_S_innerlegback, ACT_S_innerlegback_entry, ACT_S_innerlegback, ACT_S_innerlegback);
fsm1.state_def(STATE_S_innerlegswing, ACT_S_innerlegswing_entry, ACT_S_innerlegswing, ACT_S_innerlegswing);
fsm1.state_def(STATE_S_stop, ACT_S_stop, ACT_S_stop, ACT_S_stop);

fsm1.print_state_transition_matrix();

// define the transitions
// a transition needs to be defined for every state for every input
// for the sensors which dont make a difference it should stay in the same state

fsm1.trans_def(STATE_S_innerlegfront, COND_S_inlegfront_to_inlegstance, STATE_S_innerlegstance);
fsm1.trans_def(STATE_S_innerlegstance, COND_S_inlegstance_to_inlegback, STATE_S_innerlegback);
fsm1.trans_def(STATE_S_innerlegback, COND_S_inlegback_to_inlegswing, STATE_S_innerlegswing);
fsm1.trans_def(STATE_S_innerlegswing, COND_S_inlegswing_to_inlegfront, STATE_S_innerlegfront);

//Stop from any state based on COND
fsm1.trans_def(STATE_S_innerlegfront, COND_S_inlegfront_to_stop, STATE_S_stop);
fsm1.trans_def(STATE_S_innerlegstance, COND_S_inlegstance_to_stop, STATE_S_stop);
fsm1.trans_def(STATE_S_innerlegback, COND_S_inlegback_to_stop, STATE_S_stop);
fsm1.trans_def(STATE_S_innerlegswing, COND_S_inlegswing_to_stop, STATE_S_stop);

fsm1.print_state_transition_matrix();

// define exit state of fsm
fsm1.exit_state_def(STATE_S_stop);
fsm1.set_sensor_input_function(get_steering_input);
fsm1.set_state_communication_variable(&g_steering_fsm_state);
//set_io_float(ID_MB_FOOT_INNER_FSM_STATE, (float)g_foot_inner_fsm_state);
copy(fi_fsm, &fsm1);
}
```

Code: steering_conditions.cpp

```
#include <mb_includes.h>
#include "steering_fsm_conditions.h"
#include "global_sensors.h"
//#include <cstdlib>
//#include <iostream>

//using namespace std;

void get_steering_input(int* sensor_array, int sensor_array_length)
{
   int num_sensors = sensor_array_length;

  // this function should contain code to convert the global sensor values into
  // the specific sensory input for the hip_fsm. The sensory states possible
  // are listed in hip_fsm_sensors.h

   for (int i=0;i<num_sensors;i++)      // for the inactive sensors the
     sensor_array[i] =0;              // the values will be zero

  // currently generating random sensor input
```

```
 if (g_steering_fsm_stop_command == 1)
{}

 else
 {
  //if ((get_io_float(ID_E_LI_ABSANG)) > get_io_float(ID_P_F_PP_L_ABSANG))    // pushoff initiation condition
  if (
      (get_io_float(ID_MCFI_LEFT_HS) + get_io_float(ID_MCFI_RIGHT_HS) >= 1.0)
      &&
      (get_io_float(ID_MCH_ANGLE)) < get_io_float(ID_P_S_NULL_S_TANG)
      )
  {
    sensor_array[COND_S_inlegfront_to_inlegstance] = 1;
  }

  //if (get_io_float(ID_MCFO_LEFT_HS) + get_io_float(ID_MCFO_RIGHT_HS) >= 1.0)
  if ((get_io_float(ID_MCH_ANGLE)) < -get_io_float(ID_P_S_NULL_S_TANG))
  {
    sensor_array[COND_S_inlegstance_to_inlegback] = 1;
  }

  if (
      (get_io_float(ID_MCFO_LEFT_HS) + get_io_float(ID_MCFO_RIGHT_HS) >= 1.0)
      &&
      (get_io_float(ID_MCH_ANGLE)) > -get_io_float(ID_P_S_NULL_S_TANG)
      )

  {
    sensor_array[COND_S_inlegback_to_inlegswing] = 1;
  }

  //  if (get_io_float(ID_MCFO_LEFT_HS) + get_io_float(ID_MCFO_RIGHT_HS) >= 1.0)
  if ((get_io_float(ID_MCH_ANGLE)) > get_io_float(ID_P_S_NULL_S_TANG))
  {
    sensor_array[COND_S_inlegswing_to_inlegfront] = 1;
  }


 //if (get_io_float(ID_UI_BUTTONS) != 0.0) //The UI_Button seems to be set to non zero giving exit 4/5/2010
  if (0)
  {
    sensor_array[COND_S_inlegfront_to_stop] = 1;
  }

  // if (get_io_float(ID_UI_BUTTONS) != 0.0) //The UI_Button seems to be set to non zero giving exit 4/5/2010
  if (0)
  {
    sensor_array[COND_S_inlegstance_to_stop] = 1;
  }

  // if (get_io_float(ID_UI_BUTTONS) != 0.0) //The UI_Button seems to be set to non zero giving exit 4/5/2010
  if (0)
  {
    sensor_array[COND_S_inlegback_to_stop] = 1;
  }
```

```
  // if (get_io_float(ID_UI_BUTTONS) != 0.0) //The UI_Button seems to be set to non zero giving exit 4/5/2010
  if (0)
  {
    sensor_array[COND_S_inlegswing_to_stop] = 1;
  }


 }

// cout << "->hip_fsm: getting sensor input " << current_sensor_reading << endl;




  //  sensor_array[current_sensor_reading] = 1;

}
```