**NUKE.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Inverse Kinematics for 4/6 legged bots using 3DOF lizard legs

\*

\* Auto-Generated by NUKE!

\* http://arbotix.googlecode.com

\*

\* FRONT VIEW ^ ==0 0==

\* /\\_\_\_/\ | | 0==[\_\_\_]==0 |

\* / \ Z | |

\*

\* TOP VIEW

\* \ / ^

\* \\_\_\_\_\_/ |

\* \_\_\_| |\_\_\_ X

\* |\_\_\_\_\_|

\* / \ Y->

\* / \

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef NUKE

#define NUKE

#define LEG\_COUNT 6

/\* Body

\* We assume 4 legs are on the corners of a box defined by X\_COXA x Y\_COXA

\* Middle legs for a hexapod can be different Y, but should be halfway in X

\*/

#define X\_COXA 60 // MM between front and back legs /2

#define Y\_COXA 60 // MM between front/back legs /2

#define M\_COXA 100 // MM between two middle legs /2

/\* Legs \*/

#define L\_COXA 52 // MM distance from coxa servo to femur servo

#define L\_FEMUR 82 // MM distance from femur servo to tibia servo

#define L\_TIBIA 140 // MM distance from tibia servo to foot

/\* Servo IDs \*/

#define RM\_TIBIA 18

#define RF\_COXA 2

#define LR\_TIBIA 11

#define LF\_FEMUR 3

#define RF\_TIBIA 6

#define RM\_FEMUR 16

#define RM\_COXA 14

#define RR\_COXA 8

#define LF\_TIBIA 5

#define LF\_COXA 1

#define LR\_FEMUR 9

#define RR\_FEMUR 10

#define LM\_TIBIA 17

#define RF\_FEMUR 4

#define LM\_FEMUR 15

#define RR\_TIBIA 12

#define LM\_COXA 13

#define LR\_COXA 7

/\* A leg position request (output of body calcs, input to simple 3dof solver). \*/

typedef struct{

int x;

int y;

int z;

float r;

} ik\_req\_t;

/\* Servo ouptut values (output of 3dof leg solver). \*/

typedef struct{

int coxa;

int femur;

int tibia;

} ik\_sol\_t;

/\* Actual positions, and indices of array. \*/

extern ik\_req\_t endpoints[LEG\_COUNT];

#define RIGHT\_FRONT 0

#define RIGHT\_REAR 1

#define LEFT\_FRONT 2

#define LEFT\_REAR 3

#define RIGHT\_MIDDLE 4

#define LEFT\_MIDDLE 5

extern BioloidController bioloid;

/\* Parameters for manipulating body position \*/

extern float bodyRotX; // body roll

extern float bodyRotY; // body pitch

extern float bodyRotZ; // body rotation

extern int bodyPosX;

extern int bodyPosY;

/\* Parameters for gait manipulation \*/

extern int Xspeed;

extern int Yspeed;

extern float Rspeed;

extern int tranTime;

extern float cycleTime;

extern int stepsInCycle;

extern int liftHeight;

extern int step;

/\* Gait Engine \*/

extern int gaitLegNo[]; // order to move legs in

extern ik\_req\_t gaits[]; // gait position

/\* convert radians to a dynamixel servo offset \*/

int radToServo(float rads);

/\* select a gait pattern to use \*/

void gaitSelect(int GaitType);

#include "gaits.h"

/\* find the translation of the coxa point (x,y) in 3-space, given our rotations \*/

ik\_req\_t bodyIK(int X, int Y, int Z, int Xdisp, int Ydisp, float Zrot);

/\* given our leg offset (x,y,z) from the coxa point, calculate servo values \*/

ik\_sol\_t legIK(int X, int Y, int Z);

/\* ties all of the above together \*/

void doIK();

/\* setup the starting positions of the legs. \*/

void setupIK();

#endif

**NUKE.CPP**

#include <ax12.h>

#include <BioloidController.h>

#include <Arduino.h>

#include <math.h>

#include "nuke.h"

/\* min and max positions for each servo \*/

int mins[] = {222, 225, 159, 164, 279, 158, 223, 229, 159, 156, 272, 155, 226, 233, 158, 157, 271, 157};

int maxs[] = {790, 792, 855, 862, 857, 747, 788, 794, 859, 857, 860, 747, 789, 789, 858, 860, 859, 743};

/\* IK Engine \*/

BioloidController bioloid = BioloidController(1000000);

ik\_req\_t endpoints[LEG\_COUNT];

float bodyRotX = 0; // body roll (rad)

float bodyRotY = 0; // body pitch (rad)

float bodyRotZ = 0; // body rotation (rad)

int bodyPosX = 0; // body offset (mm)

int bodyPosY = 0; // body offset (mm)

int Xspeed; // forward speed (mm/s)

int Yspeed; // sideward speed (mm/s)

float Rspeed; // rotation speed (rad/s)

/\* Gait Engine \*/

int gaitLegNo[LEG\_COUNT]; // order to step through legs

ik\_req\_t gaits[LEG\_COUNT]; // gait engine output

int pushSteps; // how much of the cycle we are on the ground

int stepsInCycle; // how many steps in this cycle

int step; // current step

int tranTime;

int liftHeight;

float cycleTime; // cycle time in seconds (adjustment from speed to step-size)

/\* Setup the starting positions of the legs. \*/

void setupIK(){

endpoints[RIGHT\_FRONT].x = 52;

endpoints[RIGHT\_FRONT].y = 118;

endpoints[RIGHT\_FRONT].z = 97;

endpoints[RIGHT\_REAR].x = -52;

endpoints[RIGHT\_REAR].y = 118;

endpoints[RIGHT\_REAR].z = 97;

endpoints[RIGHT\_MIDDLE].x = 0;

endpoints[RIGHT\_MIDDLE].y = 118;

endpoints[RIGHT\_MIDDLE].z = 97;

endpoints[LEFT\_MIDDLE].x = 0;

endpoints[LEFT\_MIDDLE].y = -118;

endpoints[LEFT\_MIDDLE].z = 97;

endpoints[LEFT\_FRONT].x = 52;

endpoints[LEFT\_FRONT].y = -118;

endpoints[LEFT\_FRONT].z = 97;

endpoints[LEFT\_REAR].x = -52;

endpoints[LEFT\_REAR].y = -118;

endpoints[LEFT\_REAR].z = 97;

liftHeight = 35;

stepsInCycle = 1;

step = 0;

}

#include "gaits.h"

/\* Convert radians to servo position offset. \*/

int radToServo(float rads){

float val = (rads\*100)/51 \* 100;

return (int) val;

}

/\* Body IK solver: compute where legs should be. \*/

ik\_req\_t bodyIK(int X, int Y, int Z, int Xdisp, int Ydisp, float Zrot){

ik\_req\_t ans;

float cosB = cos(bodyRotX);

float sinB = sin(bodyRotX);

float cosG = cos(bodyRotY);

float sinG = sin(bodyRotY);

float cosA = cos(bodyRotZ+Zrot);

float sinA = sin(bodyRotZ+Zrot);

int totalX = X + Xdisp + bodyPosX;

int totalY = Y + Ydisp + bodyPosY;

ans.x = totalX - int(totalX\*cosG\*cosA + totalY\*sinB\*sinG\*cosA + Z\*cosB\*sinG\*cosA - totalY\*cosB\*sinA + Z\*sinB\*sinA) + bodyPosX;

ans.y = totalY - int(totalX\*cosG\*sinA + totalY\*sinB\*sinG\*sinA + Z\*cosB\*sinG\*sinA + totalY\*cosB\*cosA - Z\*sinB\*cosA) + bodyPosY;

ans.z = Z - int(-totalX\*sinG + totalY\*sinB\*cosG + Z\*cosB\*cosG);

return ans;

}

/\* Simple 3dof leg solver. X,Y,Z are the length from the Coxa rotate to the endpoint. \*/

ik\_sol\_t legIK(int X, int Y, int Z){

ik\_sol\_t ans;

// first, make this a 2DOF problem... by solving coxa

ans.coxa = radToServo(atan2(X,Y));

long trueX = sqrt(sq((long)X)+sq((long)Y)) - L\_COXA;

long im = sqrt(sq((long)trueX)+sq((long)Z)); // length of imaginary leg

// get femur angle above horizon...

float q1 = -atan2(Z,trueX);

long d1 = sq(L\_FEMUR)-sq(L\_TIBIA)+sq(im);

long d2 = 2\*L\_FEMUR\*im;

float q2 = acos((float)d1/(float)d2);

ans.femur = radToServo(q1+q2);

// and tibia angle from femur...

d1 = sq(L\_FEMUR)-sq(im)+sq(L\_TIBIA);

d2 = 2\*L\_TIBIA\*L\_FEMUR;

ans.tibia = radToServo(acos((float)d1/(float)d2)-1.57);

return ans;

}

void doIK(){

int servo;

ik\_req\_t req, gait;

ik\_sol\_t sol;

gaitSetup();

// right front leg

gait = gaitGen(RIGHT\_FRONT);

req = bodyIK(endpoints[RIGHT\_FRONT].x+gait.x, endpoints[RIGHT\_FRONT].y+gait.y, endpoints[RIGHT\_FRONT].z+gait.z, X\_COXA, Y\_COXA, gait.r);

sol = legIK(endpoints[RIGHT\_FRONT].x+req.x+gait.x,endpoints[RIGHT\_FRONT].y+req.y+gait.y,endpoints[RIGHT\_FRONT].z+req.z+gait.z);

servo = 368 + sol.coxa;

if(servo < maxs[RF\_COXA-1] && servo > mins[RF\_COXA-1])

bioloid.setNextPose(RF\_COXA, servo);

else{

Serial.print("RF\_COXA FAIL: ");

Serial.println(servo);

}

servo = 524 + sol.femur;

if(servo < maxs[RF\_FEMUR-1] && servo > mins[RF\_FEMUR-1])

bioloid.setNextPose(RF\_FEMUR, servo);

else{

Serial.print("RF\_FEMUR FAIL: ");

Serial.println(servo);

}

servo = 354 + sol.tibia;

if(servo < maxs[RF\_TIBIA-1] && servo > mins[RF\_TIBIA-1])

bioloid.setNextPose(RF\_TIBIA, servo);

else{

Serial.print("RF\_TIBIA FAIL: ");

Serial.println(servo);

}

// right rear leg

gait = gaitGen(RIGHT\_REAR);

req = bodyIK(endpoints[RIGHT\_REAR].x+gait.x,endpoints[RIGHT\_REAR].y+gait.y, endpoints[RIGHT\_REAR].z+gait.z, -X\_COXA, Y\_COXA, gait.r);

sol = legIK(-endpoints[RIGHT\_REAR].x-req.x-gait.x,endpoints[RIGHT\_REAR].y+req.y+gait.y,endpoints[RIGHT\_REAR].z+req.z+gait.z);

servo = 656 - sol.coxa;

if(servo < maxs[RR\_COXA-1] && servo > mins[RR\_COXA-1])

bioloid.setNextPose(RR\_COXA, servo);

else{

Serial.print("RR\_COXA FAIL: ");

Serial.println(servo);

}

servo = 524 + sol.femur;

if(servo < maxs[RR\_FEMUR-1] && servo > mins[RR\_FEMUR-1])

bioloid.setNextPose(RR\_FEMUR, servo);

else{

Serial.print("RR\_FEMUR FAIL: ");

Serial.println(servo);

}

servo = 354 + sol.tibia;

if(servo < maxs[RR\_TIBIA-1] && servo > mins[RR\_TIBIA-1])

bioloid.setNextPose(RR\_TIBIA, servo);

else{

Serial.print("RR\_TIBIA FAIL: ");

Serial.println(servo);

}

// left front leg

gait = gaitGen(LEFT\_FRONT);

req = bodyIK(endpoints[LEFT\_FRONT].x+gait.x,endpoints[LEFT\_FRONT].y+gait.y, endpoints[LEFT\_FRONT].z+gait.z, X\_COXA, -Y\_COXA, gait.r);

sol = legIK(endpoints[LEFT\_FRONT].x+req.x+gait.x,-endpoints[LEFT\_FRONT].y-req.y-gait.y,endpoints[LEFT\_FRONT].z+req.z+gait.z);

servo = 656 - sol.coxa;

if(servo < maxs[LF\_COXA-1] && servo > mins[LF\_COXA-1])

bioloid.setNextPose(LF\_COXA, servo);

else{

Serial.print("LF\_COXA FAIL: ");

Serial.println(servo);

}

servo = 500 - sol.femur;

if(servo < maxs[LF\_FEMUR-1] && servo > mins[LF\_FEMUR-1])

bioloid.setNextPose(LF\_FEMUR, servo);

else{

Serial.print("LF\_FEMUR FAIL: ");

Serial.println(servo);

}

servo = 670 - sol.tibia;

if(servo < maxs[LF\_TIBIA-1] && servo > mins[LF\_TIBIA-1])

bioloid.setNextPose(LF\_TIBIA, servo);

else{

Serial.print("LF\_TIBIA FAIL: ");

Serial.println(servo);

}

// left rear leg

gait = gaitGen(LEFT\_REAR);

req = bodyIK(endpoints[LEFT\_REAR].x+gait.x,endpoints[LEFT\_REAR].y+gait.y, endpoints[LEFT\_REAR].z+gait.z, -X\_COXA, -Y\_COXA, gait.r);

sol = legIK(-endpoints[LEFT\_REAR].x-req.x-gait.x,-endpoints[LEFT\_REAR].y-req.y-gait.y,endpoints[LEFT\_REAR].z+req.z+gait.z);

servo = 368 + sol.coxa;

if(servo < maxs[LR\_COXA-1] && servo > mins[LR\_COXA-1])

bioloid.setNextPose(LR\_COXA, servo);

else{

Serial.print("LR\_COXA FAIL: ");

Serial.println(servo);

}

servo = 500 - sol.femur;

if(servo < maxs[LR\_FEMUR-1] && servo > mins[LR\_FEMUR-1])

bioloid.setNextPose(LR\_FEMUR, servo);

else{

Serial.print("LR\_FEMUR FAIL: ");

Serial.println(servo);

}

servo = 670 - sol.tibia;

if(servo < maxs[LR\_TIBIA-1] && servo > mins[LR\_TIBIA-1])

bioloid.setNextPose(LR\_TIBIA, servo);

else{

Serial.print("LR\_TIBIA FAIL: ");

Serial.println(servo);

}

// right middle leg

gait = gaitGen(RIGHT\_MIDDLE);

req = bodyIK(endpoints[RIGHT\_MIDDLE].x+gait.x,endpoints[RIGHT\_MIDDLE].y+gait.y, endpoints[RIGHT\_MIDDLE].z+gait.z, 0, Y\_COXA, gait.r);

sol = legIK(endpoints[RIGHT\_MIDDLE].x+req.x+gait.x,endpoints[RIGHT\_MIDDLE].y+req.y+gait.y,endpoints[RIGHT\_MIDDLE].z+req.z+gait.z);

servo = 512 + sol.coxa;

if(servo < maxs[RM\_COXA-1] && servo > mins[RM\_COXA-1])

bioloid.setNextPose(RM\_COXA, servo);

else{

Serial.print("RM\_COXA FAIL: ");

Serial.println(servo);

}

servo = 524 + sol.femur;

if(servo < maxs[RM\_FEMUR-1] && servo > mins[RM\_FEMUR-1])

bioloid.setNextPose(RM\_FEMUR, servo);

else{

Serial.print("RM\_FEMUR FAIL: ");

Serial.println(servo);

}

servo = 354 + sol.tibia;

if(servo < maxs[RM\_TIBIA-1] && servo > mins[RM\_TIBIA-1])

bioloid.setNextPose(RM\_TIBIA, servo);

else{

Serial.print("RM\_TIBIA FAIL: ");

Serial.println(servo);

}

// left middle leg

gait = gaitGen(LEFT\_MIDDLE);

req = bodyIK(endpoints[LEFT\_MIDDLE].x+gait.x,endpoints[LEFT\_MIDDLE].y+gait.y, endpoints[LEFT\_MIDDLE].z+gait.z, 0, -Y\_COXA, gait.r);

sol = legIK(endpoints[LEFT\_MIDDLE].x+req.x+gait.x,-endpoints[LEFT\_MIDDLE].y-req.y-gait.y,endpoints[LEFT\_MIDDLE].z+req.z+gait.z);

servo = 512 - sol.coxa;

if(servo < maxs[LM\_COXA-1] && servo > mins[LM\_COXA-1])

bioloid.setNextPose(LM\_COXA, servo);

else{

Serial.print("LM\_COXA FAIL: ");

Serial.println(servo);

}

servo = 500 - sol.femur;

if(servo < maxs[LM\_FEMUR-1] && servo > mins[LM\_FEMUR-1])

bioloid.setNextPose(LM\_FEMUR, servo);

else{

Serial.print("LM\_FEMUR FAIL: ");

Serial.println(servo);

}

servo = 670 - sol.tibia;

if(servo < maxs[LM\_TIBIA-1] && servo > mins[LM\_TIBIA-1])

bioloid.setNextPose(LM\_TIBIA, servo);

else{

Serial.print("LM\_TIBIA FAIL: ");

Serial.println(servo);

}

step = (step+1)%stepsInCycle;

}

**GAITS.H**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Gaits Auto-Generated by NUKE!

\* http://arbotix.googlecode.com

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This is a bit funky -- NEVER INCLUDE gaits.h in your file! \*/

#ifndef GAIT\_H

#define GAIT\_H

/\* find the translation of the endpoint (x,y,z) given our gait parameters \*/

extern ik\_req\_t (\*gaitGen)(int leg);

extern void (\*gaitSetup)();

/\* ripple gaits move one leg at a time

\* for going forward, or turning left/right

\*/

#define RIPPLE 0

#define RIPPLE\_SMOOTH 3

#define AMBLE 4

#define AMBLE\_SMOOTH 5

/\* tripod gaits are only for hexapods \*/

#define TRIPOD 6

#define MOVING ((Xspeed > 5 || Xspeed < -5) || (Yspeed > 5 || Yspeed < -5) || (Rspeed > 0.05 || Rspeed < -0.05))

/\* Standard Transition time should be of the form (k\*BIOLOID\_FRAME\_LENGTH)-1

\* for maximal accuracy. BIOLOID\_FRAME\_LENGTH = 33ms, so good options include:

\* 32, 65, 98, etc...

\*/

#define STD\_TRANSITION 98 //98 for ax-12 hexapod, 32 for ax-18f

#else

/\* Simple calculations at the beginning of a cycle. \*/

void DefaultGaitSetup(){

// nothing!

}

/\* Simple, fast, and rough gait. Legs will make a fast triangular stroke. \*/

ik\_req\_t DefaultGaitGen(int leg){

if( MOVING ){

// are we moving?

if(step == gaitLegNo[leg]){

// leg up, middle position

gaits[leg].x = 0;

gaits[leg].y = 0;

gaits[leg].z = -liftHeight;

gaits[leg].r = 0;

}else if(((step == gaitLegNo[leg]+1) || (step == gaitLegNo[leg]-(stepsInCycle-1))) && (gaits[leg].z < 0)){

// leg down position NOTE: dutyFactor = pushSteps/StepsInCycle

gaits[leg].x = (Xspeed\*cycleTime\*pushSteps)/(2\*stepsInCycle); // travel/Cycle = speed\*cycleTime

gaits[leg].y = (Yspeed\*cycleTime\*pushSteps)/(2\*stepsInCycle); // Stride = travel/Cycle \* dutyFactor

gaits[leg].z = 0; // = speed\*cycleTime\*pushSteps/stepsInCycle

gaits[leg].r = (Rspeed\*cycleTime\*pushSteps)/(2\*stepsInCycle); // we move Stride/2 here

}else{

// move body forward

gaits[leg].x = gaits[leg].x - (Xspeed\*cycleTime)/stepsInCycle; // note calculations for Stride above

gaits[leg].y = gaits[leg].y - (Yspeed\*cycleTime)/stepsInCycle; // we have to move Stride/pushSteps here

gaits[leg].z = 0; // = speed\*cycleTime\*pushSteps/stepsInCycle\*pushSteps

gaits[leg].r = gaits[leg].r - (Rspeed\*cycleTime)/stepsInCycle; // = speed\*cycleTime/stepsInCycle

}

}else{ // stopped

gaits[leg].z = 0;

}

return gaits[leg];

}

/\* Smoother, slower gait. Legs will make a arc stroke. \*/

ik\_req\_t SmoothGaitGen(int leg){

if( MOVING ){

// are we moving?

if(step == gaitLegNo[leg]){

// leg up, halfway to middle

gaits[leg].x = gaits[leg].x/2;

gaits[leg].y = gaits[leg].y/2;

gaits[leg].z = -liftHeight/2;

gaits[leg].r = gaits[leg].r/2;

}else if((step == gaitLegNo[leg]+1) && (gaits[leg].z < 0)){

// leg up position

gaits[leg].x = 0;

gaits[leg].y = 0;

gaits[leg].z = -liftHeight;

gaits[leg].r = 0;

}else if((step == gaitLegNo[leg] + 2) && (gaits[leg].z < 0)){

// leg halfway down

gaits[leg].x = (Xspeed\*cycleTime\*pushSteps)/(4\*stepsInCycle);

gaits[leg].y = (Yspeed\*cycleTime\*pushSteps)/(4\*stepsInCycle);

gaits[leg].z = -liftHeight/2;

gaits[leg].r = (Rspeed\*cycleTime\*pushSteps)/(4\*stepsInCycle);

}else if((step == gaitLegNo[leg]+3) && (gaits[leg].z < 0)){

// leg down position NOTE: dutyFactor = pushSteps/StepsInCycle

gaits[leg].x = (Xspeed\*cycleTime\*pushSteps)/(2\*stepsInCycle); // travel/Cycle = speed\*cycleTime

gaits[leg].y = (Yspeed\*cycleTime\*pushSteps)/(2\*stepsInCycle); // Stride = travel/Cycle \* dutyFactor

gaits[leg].z = 0; // = speed\*cycleTime\*pushSteps/stepsInCycle

gaits[leg].r = (Rspeed\*cycleTime\*pushSteps)/(2\*stepsInCycle); // we move Stride/2 here

}else{

// move body forward

gaits[leg].x = gaits[leg].x - (Xspeed\*cycleTime)/stepsInCycle; // note calculations for Stride above

gaits[leg].y = gaits[leg].y - (Yspeed\*cycleTime)/stepsInCycle; // we have to move Stride/pushSteps here

gaits[leg].z = 0; // = speed\*cycleTime\*pushSteps/stepsInCycle\*pushSteps

gaits[leg].r = gaits[leg].r - (Rspeed\*cycleTime)/stepsInCycle; // = speed\*cycleTime/stepsInCycle

}

}else{ // stopped

gaits[leg].z = 0;

}

return gaits[leg];

}

int currentGait = -1;

void gaitSelect(int GaitType){

if(GaitType == currentGait)

return;

currentGait = GaitType;

tranTime = STD\_TRANSITION;

cycleTime = 0;

// simple ripple, 12 steps

if(GaitType == RIPPLE){

gaitGen = &DefaultGaitGen;

gaitSetup = &DefaultGaitSetup;

gaitLegNo[RIGHT\_FRONT] = 0;

gaitLegNo[LEFT\_REAR] = 2;

gaitLegNo[LEFT\_MIDDLE] = 4;

gaitLegNo[LEFT\_FRONT] = 6;

gaitLegNo[RIGHT\_REAR] = 8;

gaitLegNo[RIGHT\_MIDDLE] = 10;

pushSteps = 10;

stepsInCycle = 12;

}else if(GaitType == RIPPLE\_SMOOTH){

gaitGen = &SmoothGaitGen;

gaitSetup = &DefaultGaitSetup;

gaitLegNo[RIGHT\_FRONT] = 0;

gaitLegNo[LEFT\_REAR] = 4;

gaitLegNo[LEFT\_MIDDLE] = 8;

gaitLegNo[LEFT\_FRONT] = 12;

gaitLegNo[RIGHT\_REAR] = 16;

gaitLegNo[RIGHT\_MIDDLE] = 20;

pushSteps = 20;

stepsInCycle = 24;

tranTime = 65;

}else if(GaitType == AMBLE\_SMOOTH){

gaitGen = &SmoothGaitGen;

gaitSetup = &DefaultGaitSetup;

gaitLegNo[RIGHT\_FRONT] = 0;

gaitLegNo[LEFT\_REAR] = 0;

gaitLegNo[LEFT\_FRONT] = 4;

gaitLegNo[RIGHT\_REAR] = 4;

gaitLegNo[RIGHT\_MIDDLE] = 8;

gaitLegNo[LEFT\_MIDDLE] = 8;

pushSteps = 8;

stepsInCycle = 12;

tranTime = 65;

}else if(GaitType == AMBLE){

gaitGen = &DefaultGaitGen;

gaitSetup = &DefaultGaitSetup;

gaitLegNo[RIGHT\_FRONT] = 0;

gaitLegNo[LEFT\_REAR] = 0;

gaitLegNo[LEFT\_FRONT] = 2;

gaitLegNo[RIGHT\_REAR] = 2;

gaitLegNo[RIGHT\_MIDDLE] = 4;

gaitLegNo[LEFT\_MIDDLE] = 4;

pushSteps = 4;

stepsInCycle = 6;

}else if(GaitType == TRIPOD){

gaitGen = &DefaultGaitGen;

gaitSetup = &DefaultGaitSetup;

gaitLegNo[RIGHT\_FRONT] = 0;

gaitLegNo[LEFT\_MIDDLE] = 0;

gaitLegNo[RIGHT\_REAR] = 0;

gaitLegNo[LEFT\_FRONT] = 2;

gaitLegNo[RIGHT\_MIDDLE] = 2;

gaitLegNo[LEFT\_REAR] = 2;

pushSteps = 2;

stepsInCycle = 4;

tranTime = 65;

}

if(cycleTime == 0)

cycleTime = (stepsInCycle\*tranTime)/1000.0;

step = 0;

}

ik\_req\_t (\*gaitGen)(int leg) = &DefaultGaitGen;

void (\*gaitSetup)() = &DefaultGaitSetup;

#endif