little_big_guy_v1_1.c

```
#include
#include
#include
/* I am making a few assumptions for this skellycode. Below are the
assumptions;
* 1) There are exactly 4 motor ports on both big and little guy.
* 2) Motor ports 1 & 2 are the forward wheels, ports 3 & 4 are the rear wheels.
* 3) There are the same amnt of servos on big and little guy.
* 4) You leave my calls at the beginning of static main alone!!
* 5) For the turn functions, I use rapid turn which suffers from pivot drift
(the wheel the bot turns on initially ends up at the center)
* 6) The servo movement functions pause the program while they move. If this is
not the case I have a super simple fix for that anyways :)

*Ver: a1.1
*/
//=====[GLOBAL VARIABLES]=====//
//default threshold for what is considered 'black' by the system logic
static struct pixel BLACK_THRESH;
BLACK_THRESH.r = 125;
BLACK_THRESH.g = 125;
BLACK_THRESH.b = 125;

static int SERVO_MAX = 2040; //servo max rotation used to weakly prevent self-
inflicted damage on movement calls

static int MMS_FOR_RA = 2000; //milliseconds for a 90 degree turn with standard
speed

static int DEFAULT_SPEED = 75; //standard / default velocity for the bot

//motor identifiers for semiautonomous motor control
static int FRONT_LEFT = 1;
static int FRONT_RIGHT = 2;
static int REAR_LEFT = 3;
static int REAR_RIGHT = 4;

//servo identifiers for semiautonomous claw control
static int SHOULDER_X_SERVO = 1;
static int SHOULDER_Y_SERVO = 2;
static int ELBOW_SERVO = 3;
static int FINGER_SERVO = 4;

//claw values for claw servo safety net
static int CLOSED_VALUE = 1850; // servo value when claw is closed normally
static int OPEN_VALUE = 1400; // servo value when claw is opened normally

static int X_AXIS_MIN = 0; // lowest servo value for x axis rotation of claw
static int X_AXIS_MAX = 2000; // max servo value for x axis rotation of claw

static int Y_AXIS_MIN = 500; // value at which the claw is as low as it can
```

```
safely go
static int Y_AXIS_MAX = 1500; // the max value for claw raise (the highest it
can go)

static int E_AXIS_MIN = 0; //lowest the elbow can be bent (min elbow servo val)
static int E_AXIS_MAX = 1500; // most elbow can be raised / straightened (max
elbow servo val)

//=====[MOTOR FUNCTIONS]=====//
static void mv_fwd(){ // make the bot move forward
motor(FRONT_LEFT, DEFAULT_SPEED);
motor(FRONT_RIGHT, DEFAULT_SPEED);
return;
}

static void mv_bwd(){ // make the bot move backward
motor(REAR_LEFT, 0-DEFAULT_SPEED);
motor(REAR_RIGHT, 0-DEFAULT_SPEED);
return;
}

static void turn_left(){ //make the bot make a 90 degree turn to the left
motor(FRONT_LEFT, 0-DEFAULT_SPEED);
motor(FRONT_RIGHT, DEFAULT_SPEED);
msleep(MMS_FOR_RA);
ao();
return;
}

static void turn_right(){ //make the bot make a 90 degree turn to the right
motor(FRONT_RIGHT, 0-DEFAULT_SPEED);
motor(FRONT_LEFT, DEFAULT_SPEED);
msleep(MMS_FOR_RA);
ao();
return;
}

static void stop_all(){ // stop ALL motors
ao();
}

static void stop_mtr(int motor){ // stop SPECIFIC motor
off(motor);
}

//=====[CLAW FUNCTIONS]=====//
static bool servo_safety_check(int port, int request){ //you won't use this,
its just for behind the scenes to keep the servos as safe as possible
//super simple safety logic
int ssccurrent = get_servo_position();
if((ssccurrent+request) >= SERVO_MAX){ //prematurely stop if servo is going to
exceed threshold
printf("WARNING! CALL ABORTED\nCAUSE: EXCEEDING SAFE SERVO THRESHOLD\n");
return false;
}
```

```
return true;
}

static bool x_axis_safety(int req){ //you won't use this, it's just for behind
the scenes shoulder rotation protection
int xacurrent = get_servo_position();
if((X_AXIS_MIN < xacurrent+req) && (xacurrent+req < X_AXIS_MAX)){return true;}
//good case
return false;
}

static bool y_axis_safety(int req){ //you won't use this, it's just for behind
the scenes shoulder rotation protection
int yacurrent = get_servo_position();
if((Y_AXIS_MIN < yacurrent+req) && (yacurrent+req < Y_AXIS_MAX)){return true;}
//good case
return false;
}

static bool e_axis_safety(int req){ //you won't use this, it's just for behind
the scenes elbow rotation protection
int eacurrent = get_servo_position();
if((E_AXIS_MIN < eacurrent+req) && (eacurrent+req < E_AXIS_MAX)){return true;}
//good case
return false;
}

static void turn_claw(int degrees){ //rotates the claw n 'degrees' past what it
is now (degrees == servo number thingy, positive == clockwise)
int current = get_servo_position();
if(!servo_safety_check(SHOULDER_X_SERVO, degrees)){return;} //premature
cancellation if safety check fails
if(!x_axis_safety(degrees)){return;} //premature cancellation if safety check
fails
set_servo_position(SHOULDER_X_SERVO, (current+degrees)); // move servo
return;
}

static void raise_claw(int degrees){ //changes claw position n 'degrees' past
what it is now (negative == down)
int current = get_servo_position();
if(!servo_safety_check(SHOULDER_Y_SERVO, degrees)){return;} //premature
cancelation if safety check fails
if(!y_axis_safety(degrees)){return;} //premature cancellation if safety check
fails
set_servo_position(SHOULDER_Y_SERVO, (current+degrees)); //move servo
return;
}

static void bend_elbow(int degrees){ //changes claw elbow n 'degrees' past what
it is now (negative == down)
int current = get_servo_position();
if(!servo_safety_check(ELBOW_SERVO, current+degrees)){return;} //premature
cancellation if safety check fails
if(!e_axis_safety(degrees)){return;} //premature cancellation if safety check
```

```
fails
set_servo_position(ELBOW_SERVO, (currently+degrees)); // move servo
return;
}

static void open_claw(){ //self explanatory
set_servo_position(FINGER_SERVO, OPEN_VALUE);
return;
}

static void close_claw(){ //self explanatory
set_servo_position(FINGER_SERVO, CLOSED_VALUE);
return;
}
//=====[SENSOR FUNCTIONS]=====//
bool on_black(){ // checks if CENTER PIXEL meets predefined black threshold,
returns true or false accordingly
//first find the center pixel of the camera as native wallaby.geom point2
struct
struct point2 cpx;
cpx.x = get_camera_width()/2;
cpx.y = get_camera_height()/2;

camera_update(); //update
struct pixel returned = camera_get_pixel(cpx); //get center pixel

if((BLACK_THRESH.r < returned.r) || (BLACK_THRESH.g < returned.g) ||
(BLACK_THRESH.b < returned.b)) {return false;}
else {return true;}
}
//====main
static void main(){
//setup
camera_open();
enable_servos();
printf("SETUP COMPLETE\nGOOD TO GO! :)\n");

//simple tests
for(int i=0;i<2;i++){ //fwd then bwd for 3 seconds each
mv_fwd();
msleep(3000);
stop_all();
msleep(500);
mv_bwd();
msleep(3000);
stop_all();
}
for(int x=0;x<2;x++){ //run 2 'laps', turning different directions twice
mv_fwd();
msleep(3000);
stop_all();
if(x==0){
turn_left();
turn_left();
}
```

```
else{
turn_right();
turn_right();
}
}
turn_claw(200);
msleep(500);

raise_claw(200);
msleep(500);

bend_elbow(200);
msleep(500);

open_claw();
msleep(750);

close_claw();
msleep(250);

bend_elbow(-200);
msleep(500);

raise_claw(-200);
msleep(500);

turn_claw(-200);
}
```