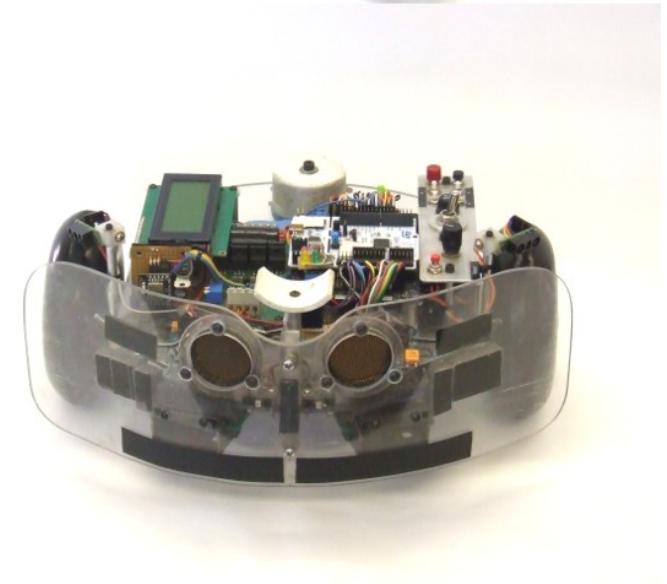
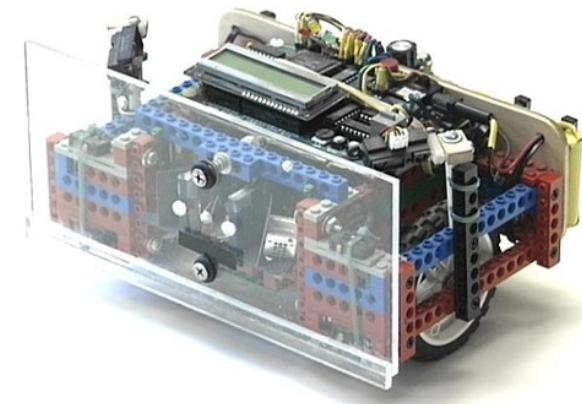
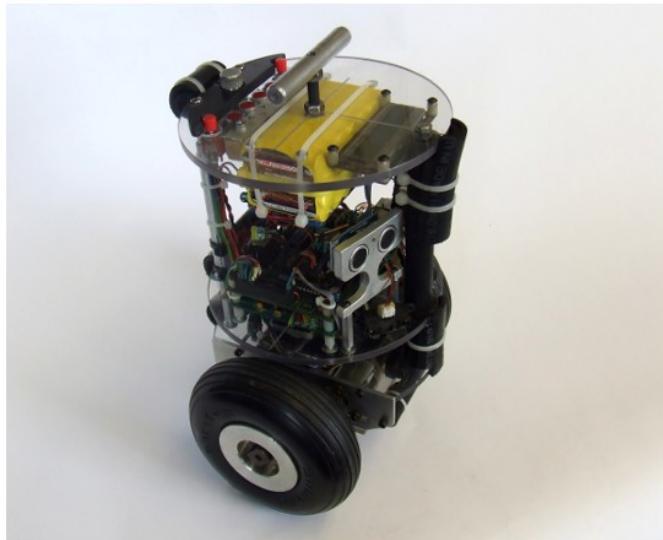


Robot Navigation

DPRG June 2021
David P. Anderson



Emergent Navigation:

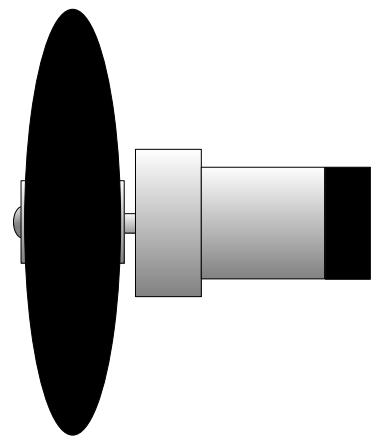
Emergent: Complex high level behavior arising from the interaction of multiple simple low level behaviors

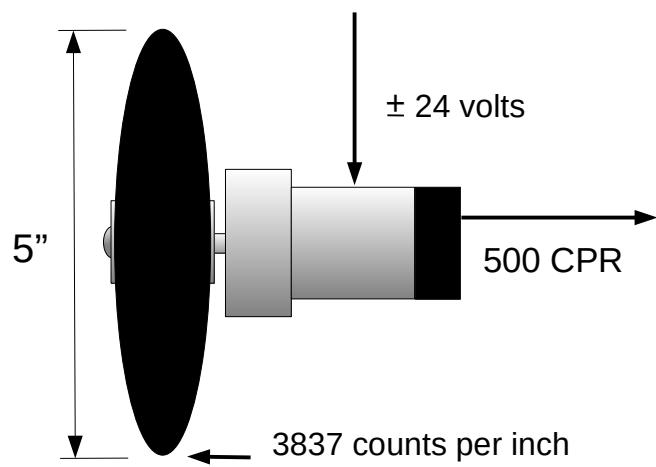
“How Waypoint Navigation is Implemented on my Robots”

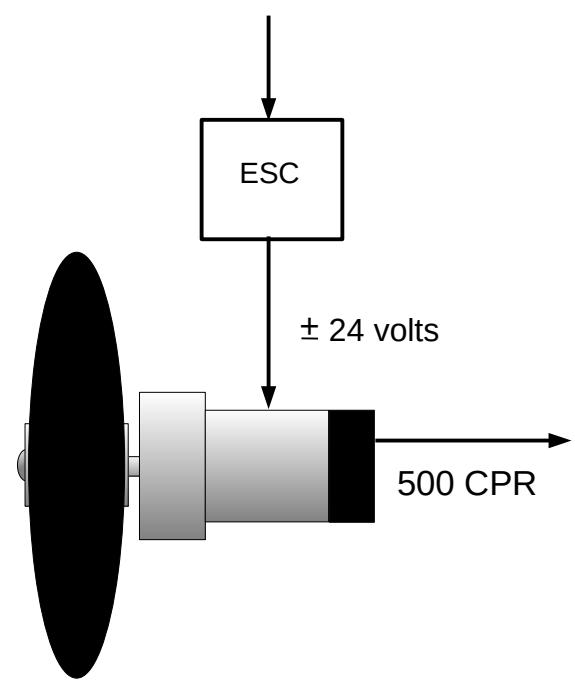
- I. Motion Control
- II. Waypoint Navigation
- III. Advanced Navigation Modes

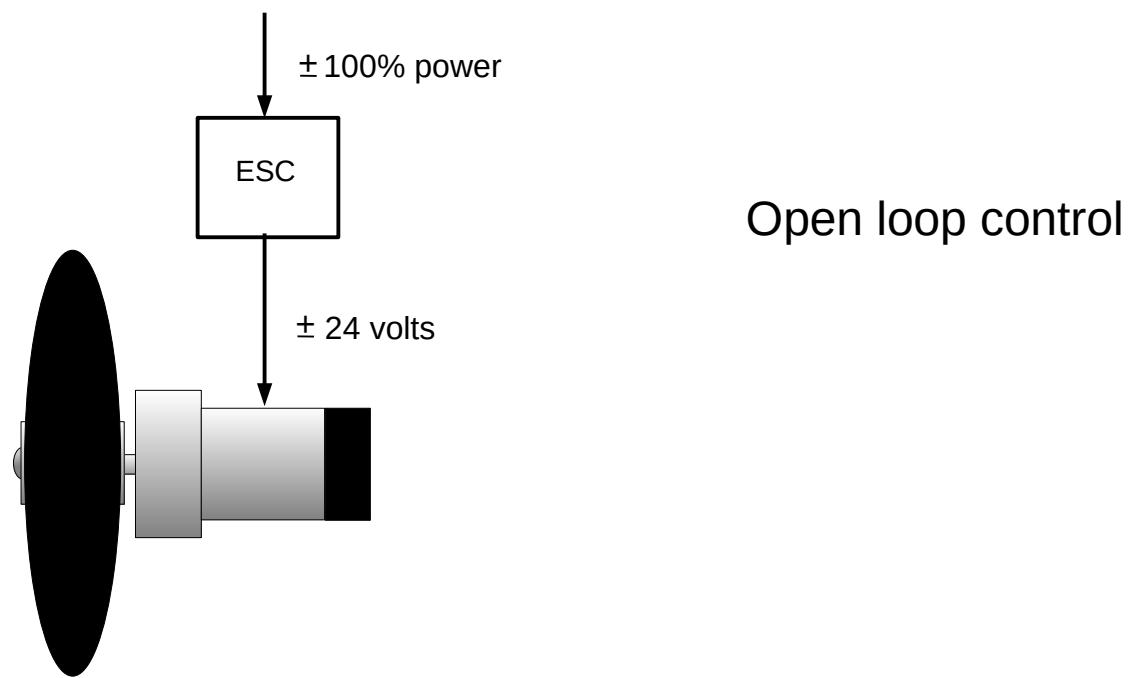
I. Motion Control

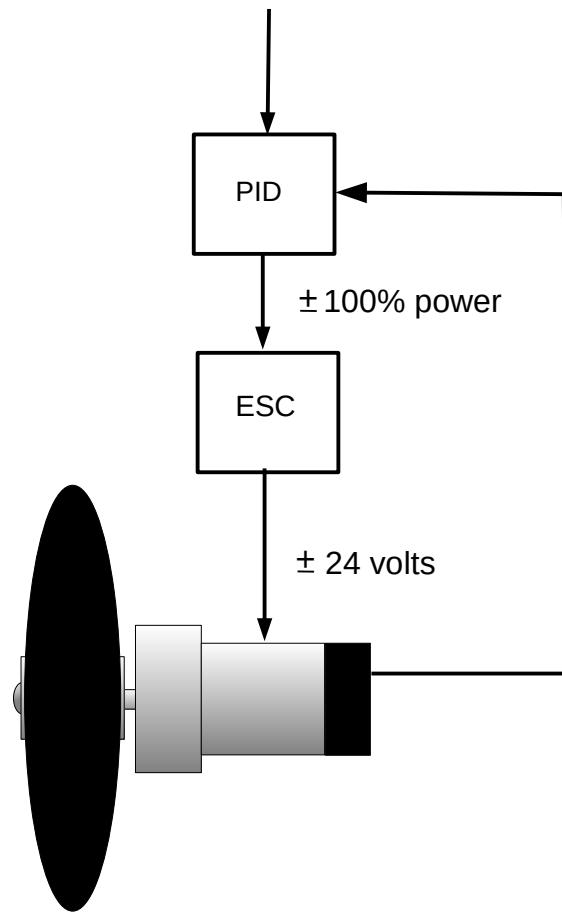
- A. Motor Control
- B. Arbiter Control Loop
- C. Example Behaviors



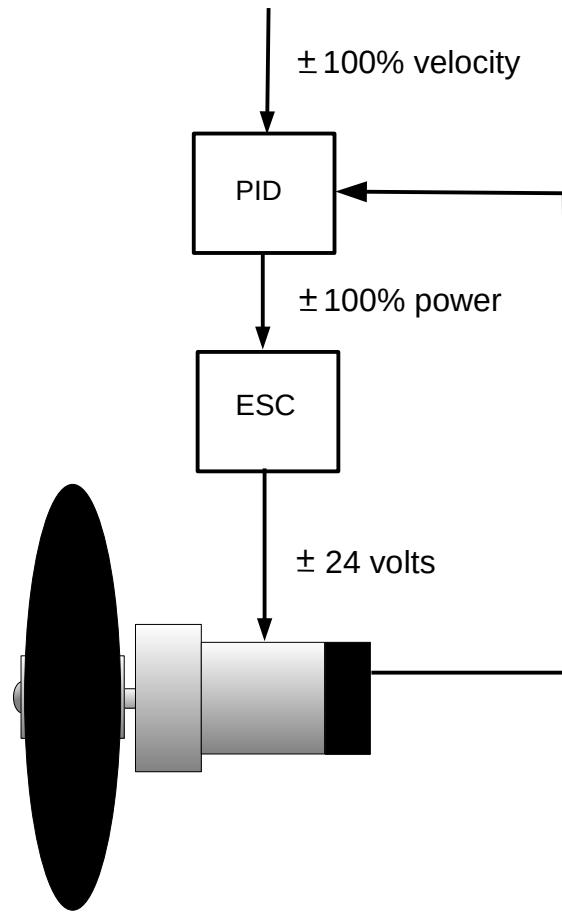




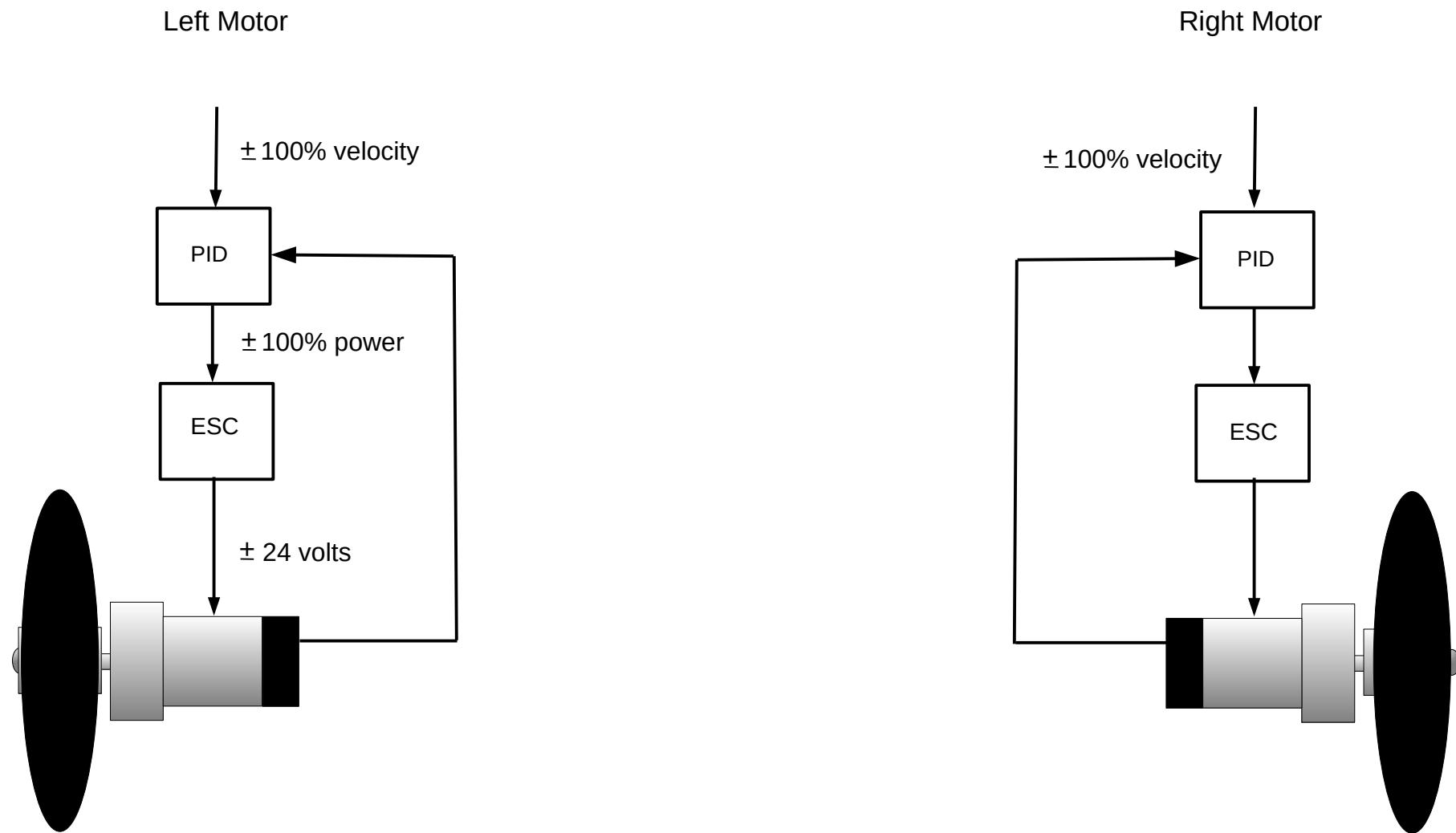




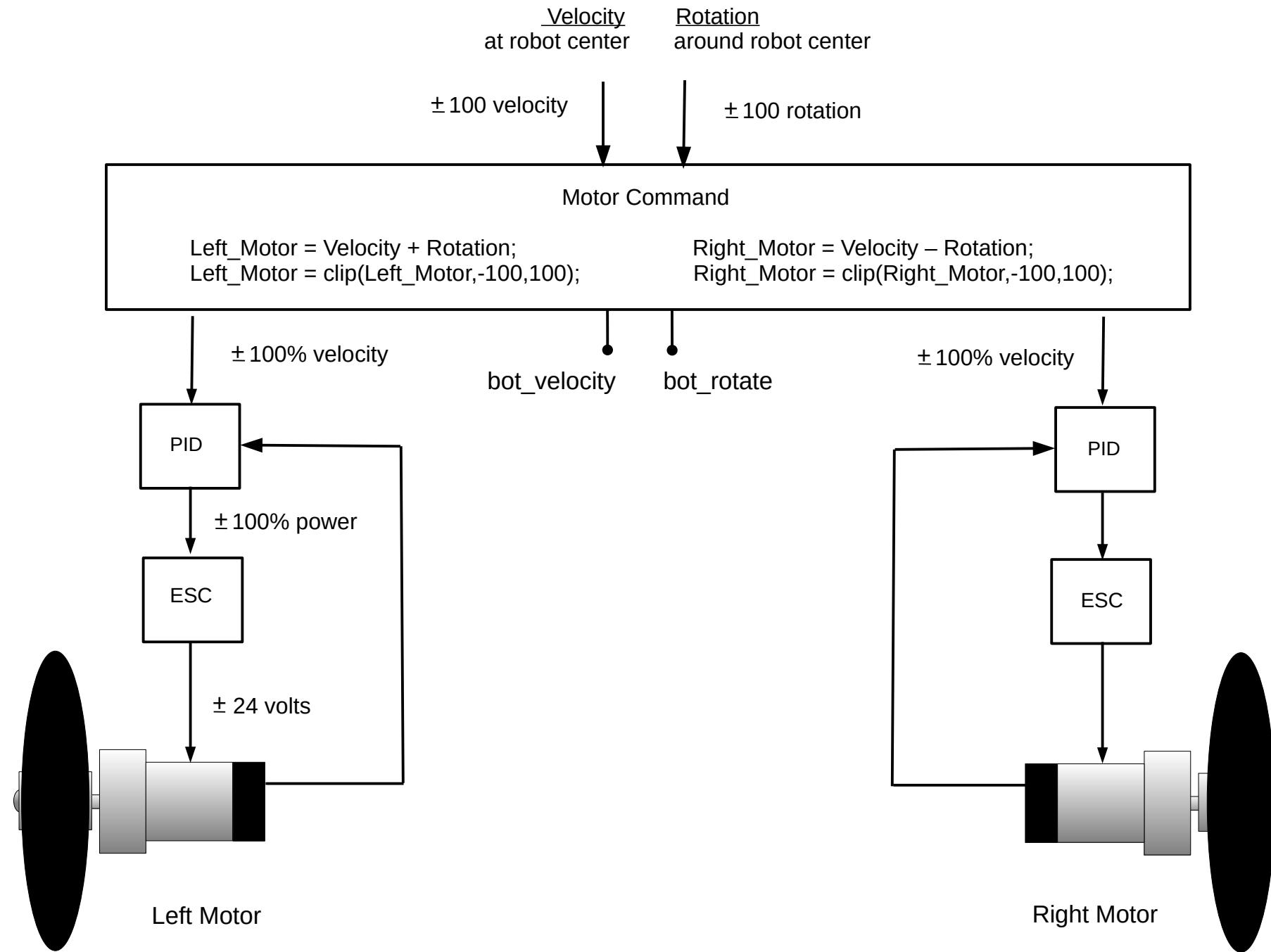
Closed loop control
25 Hz Continuous Message



Closed loop control
25 Hz Continuous Message



Motor Control Subsystem



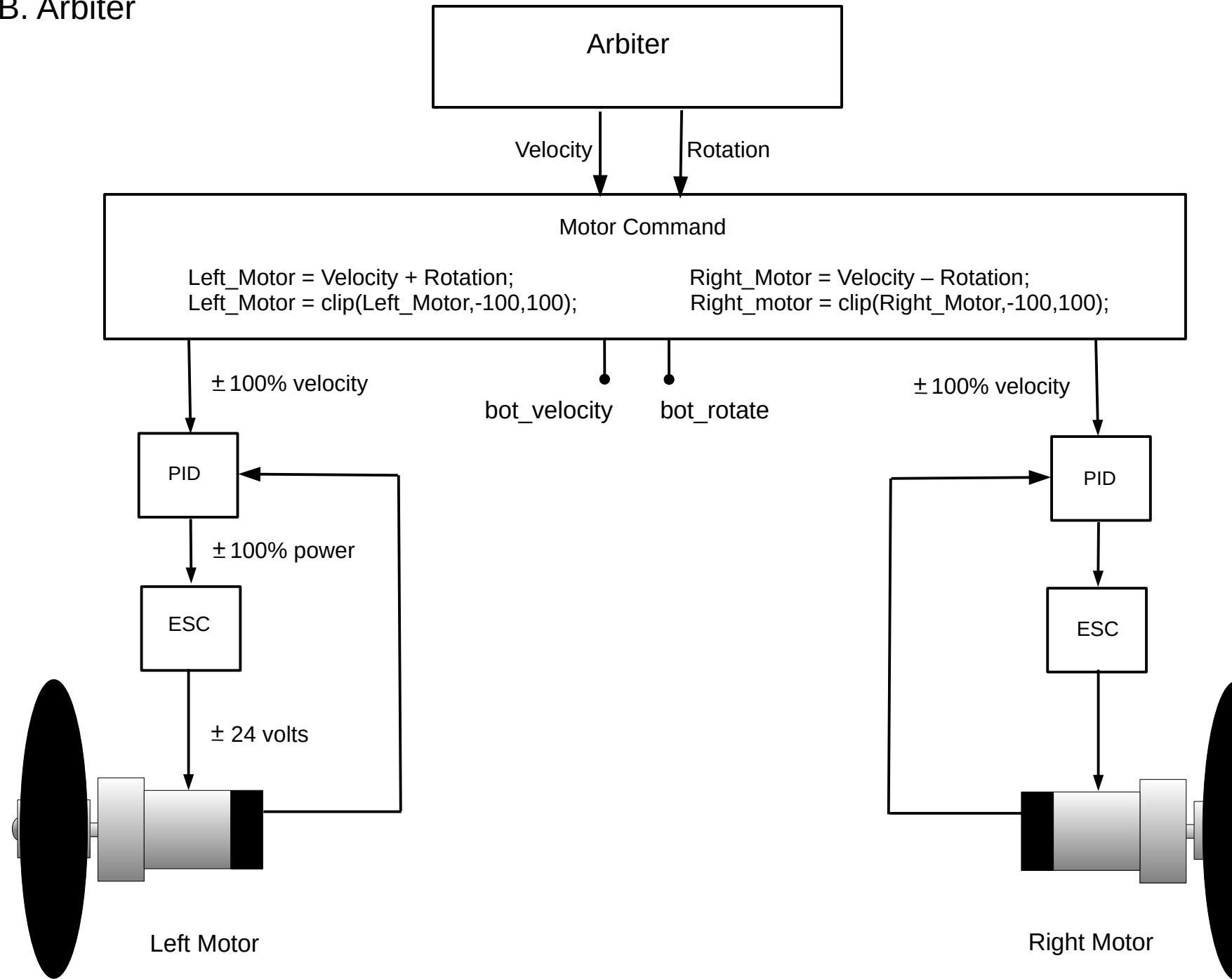
Clip() to minimum and maximum range.

```
float clip ( value, min, max )
{
    if ( value > max ) return max;
    if ( value < min ) return min;
    return value;
}
```

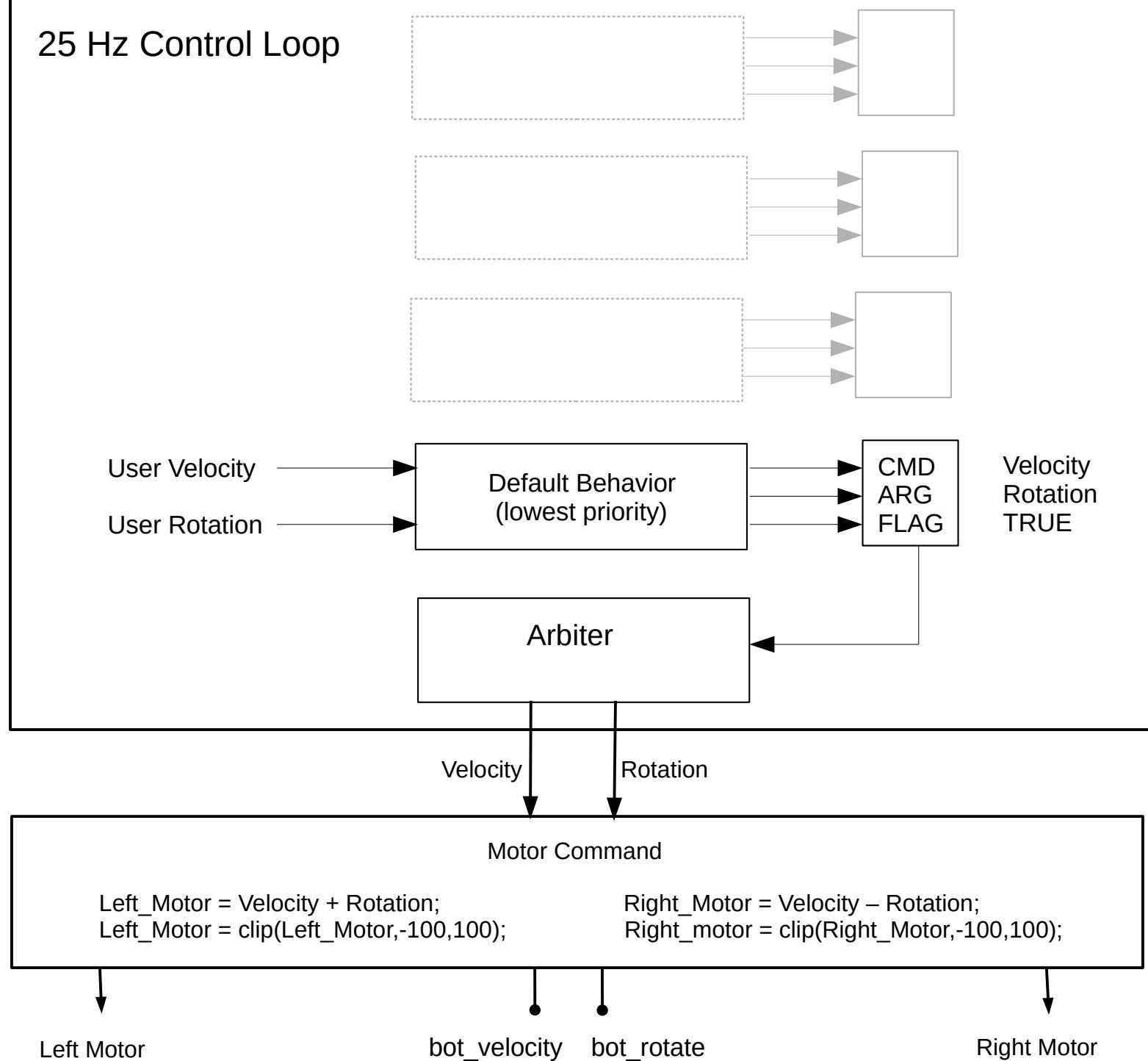
Demo 1

Live
RCAT: Velocity and Rotation

B. Arbiter



25 Hz Control Loop



```
typedef struct cmdblock CMDBLOCK;  
  
struct cmdblock  
{  
    float cmd;      // velocity  
    float arg;      // rotation  
    int flag;       // execute  
}
```

```
CMDBLOCK default;           // output block for arbiter

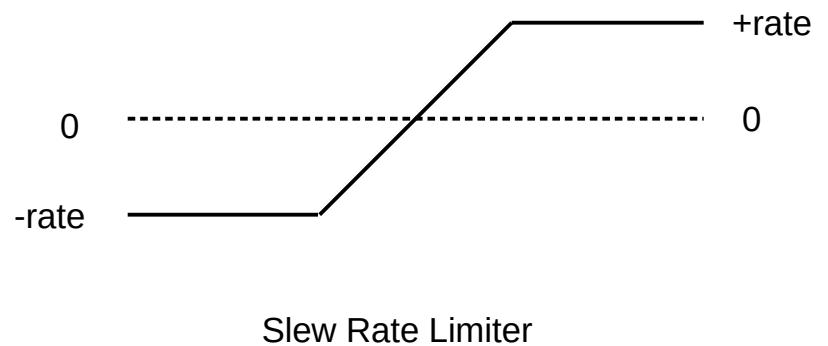
void default_behavior( void ) // 25 Hz behavior
{
    default.cmd = p_user_velocity;
    default.arg = p_user_rotate;
    default.flag = TRUE;
}
```

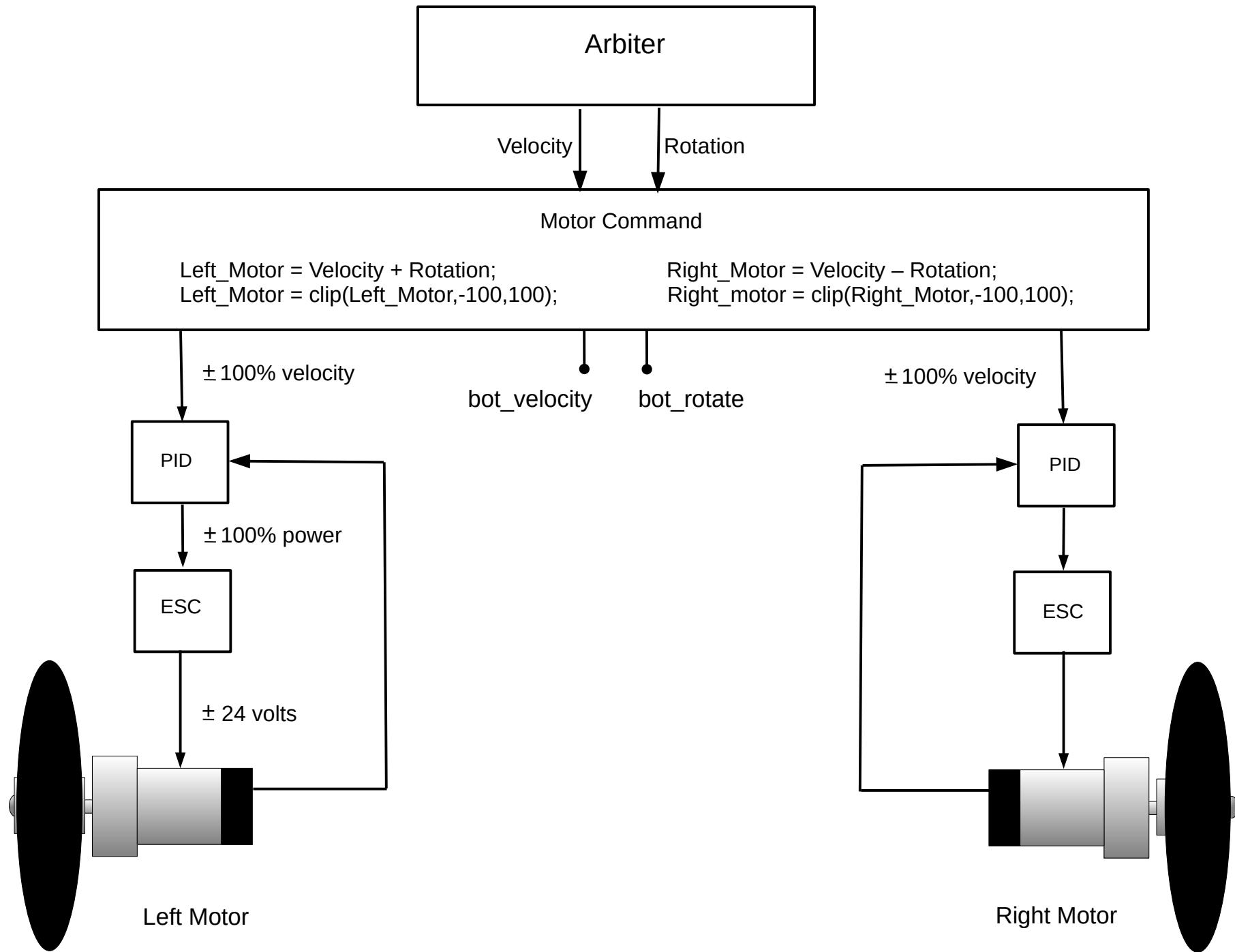
```
CMDBLOCK default;

void default_behavior ( void )
{
    default.cmd = slew_vel ( p_user_velocity, p_user_vrate );
    default.arg = slew_rot ( p_user_rotate, p_user_rrate );
    default.flag = TRUE;
}
```

```
void default_behavior ( void )
{
    default.cmd = slew_vel ( p_user_velocity, p_user_vrate );
    default.arg = slew_rot ( p_user_rotate, p_user_rrate );
    default.flag = TRUE;
}
```

```
float slew ( float from, to, rate )
{
    float dif = to - from;
    if (dif > rate) return (from + rate);
    if (dif < -rate) return (from - rate);
    return to;
}
```





```

void default_behavior (void)
{
    default.cmd = slew_vel ( p_user_velocity, p_user_vrate );
    default.arg = slew_rot ( p_user_rotate, p_user_rrate );
    default.flag = TRUE;
}

```

```

float slew ( float from, to, rate )
{
    float dif = to - from;
    if (dif > rate) return (from + rate);
    if (dif < -rate) return (from - rate);
    return to;
}

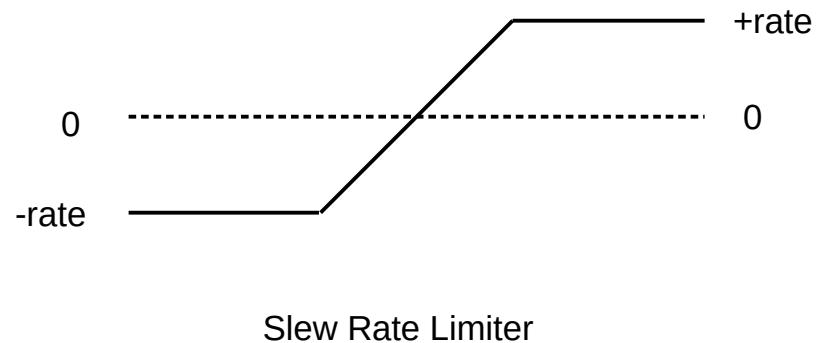
```

```

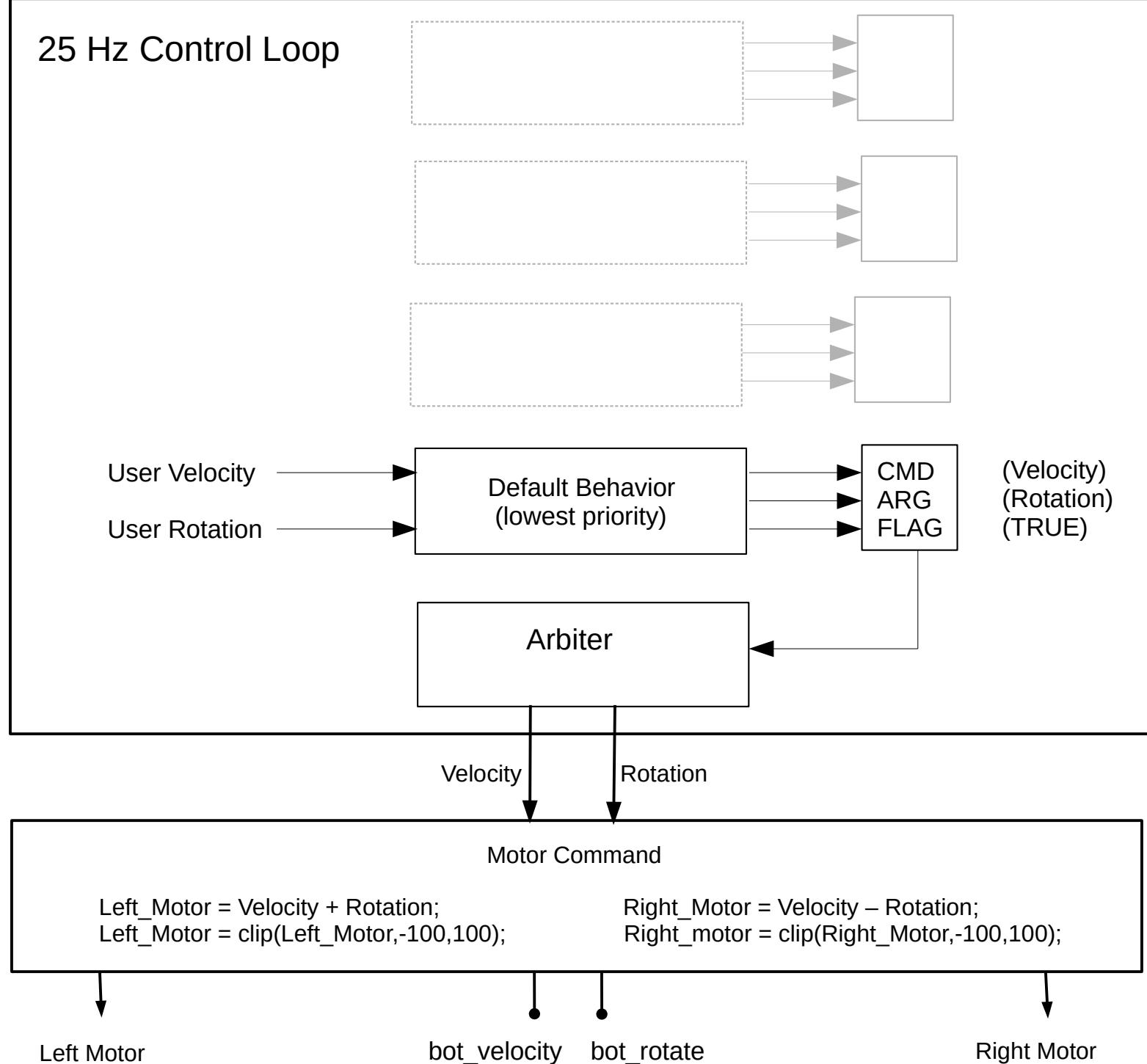
float slew_vel ( float to, rate )
{
    return slew ( bot_velocity, to, rate );
}

float slew_rot ( float to, rate )
{
    return slew ( bot_rotate, to, rate );
}

```



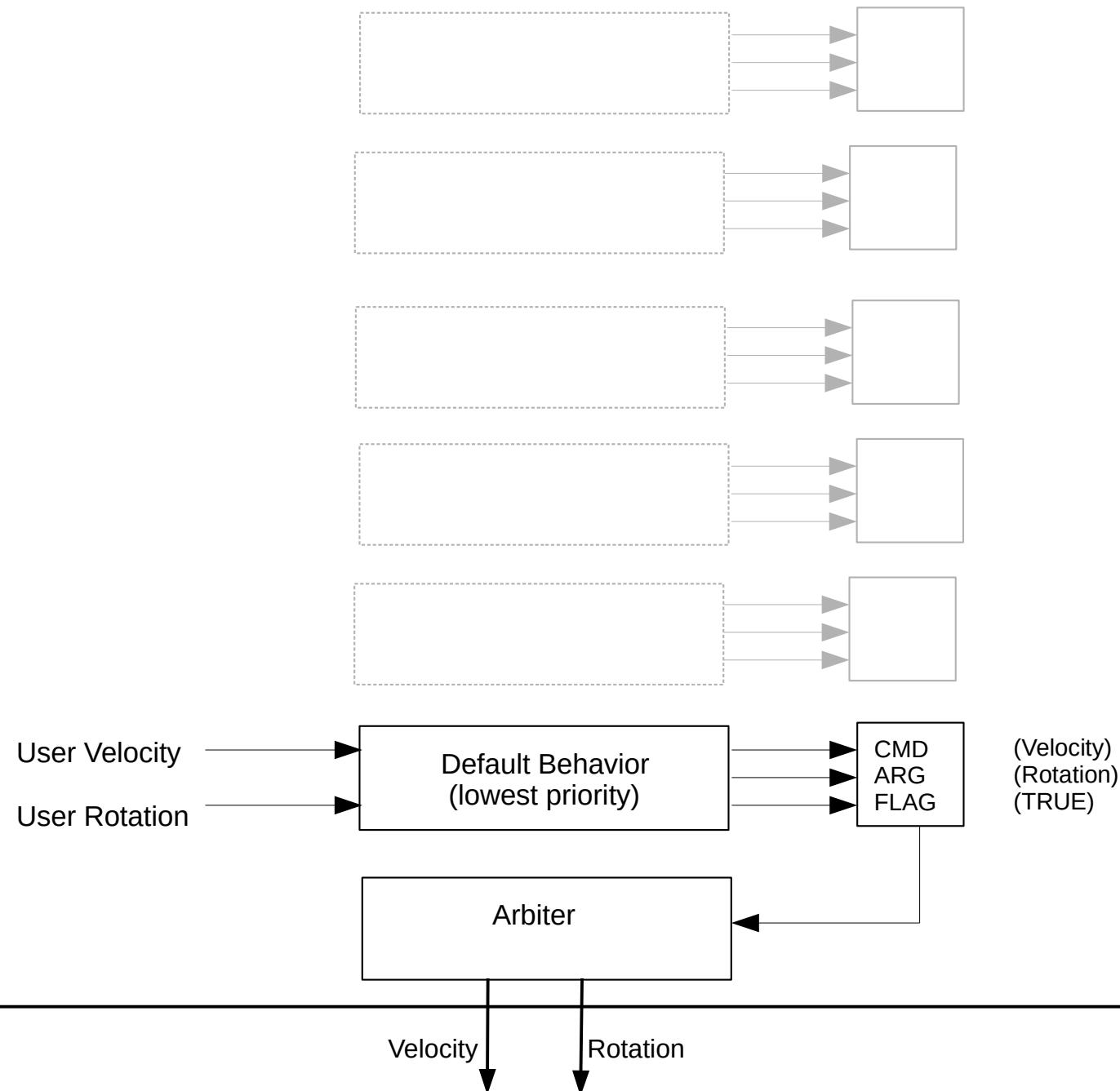
25 Hz Control Loop



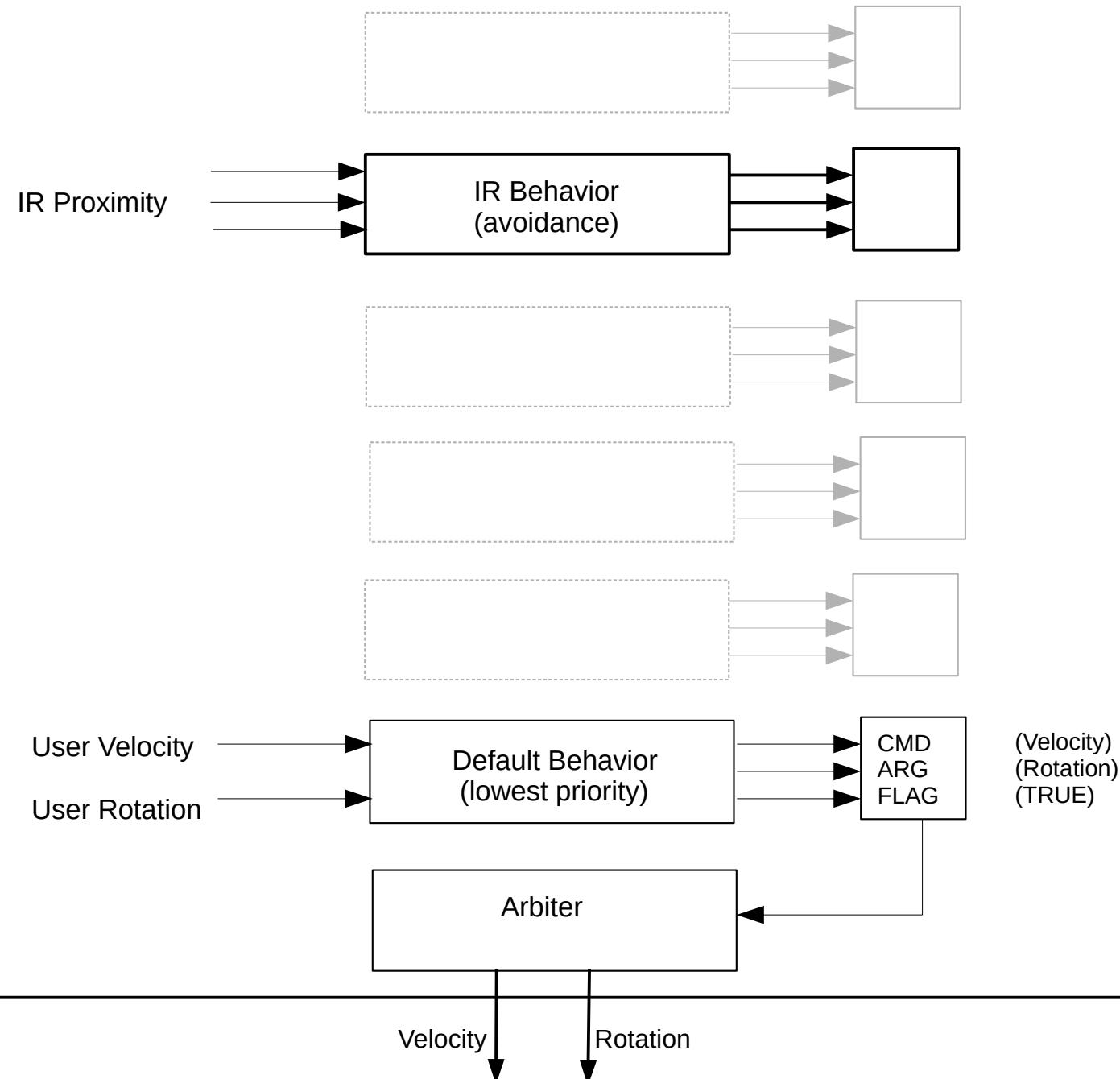
Demo 2

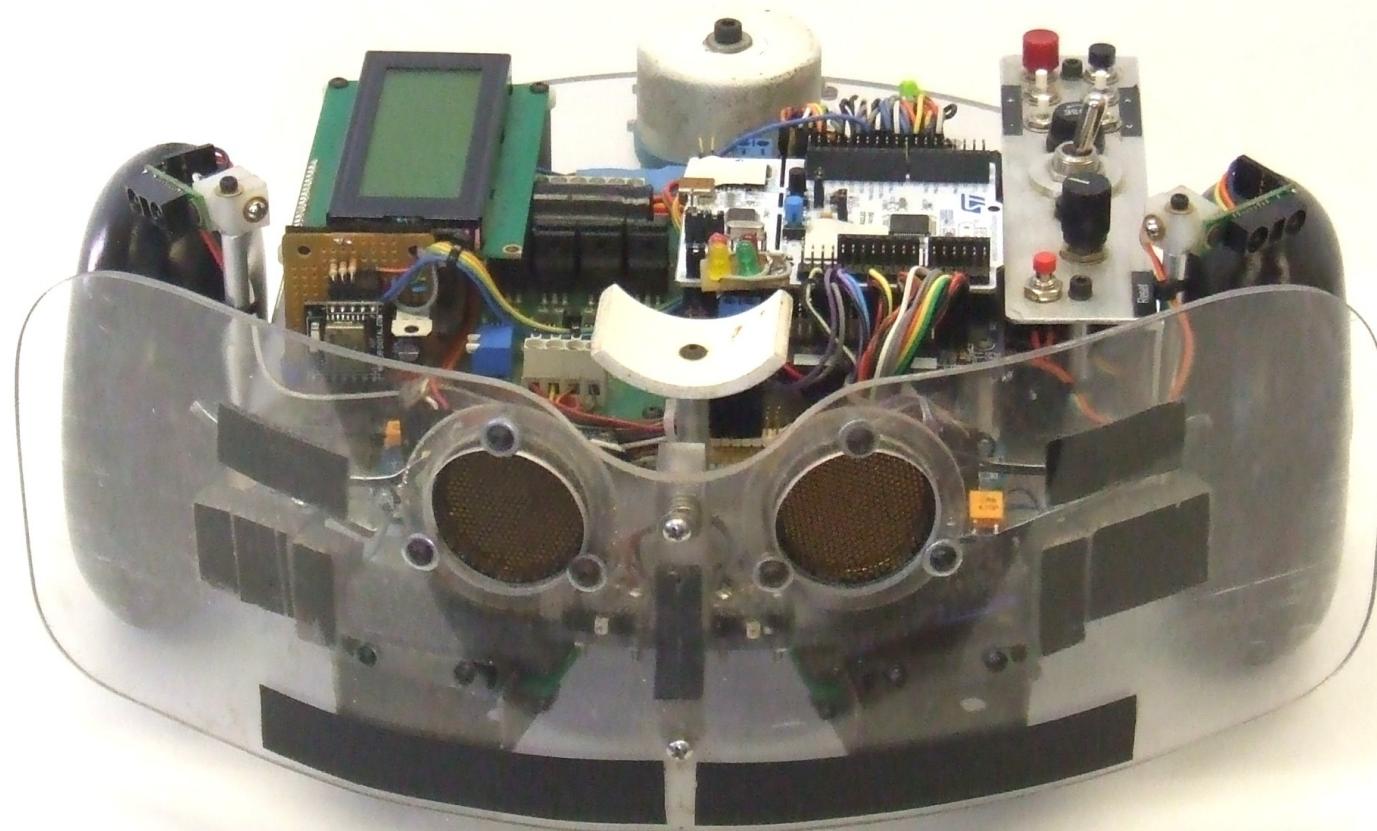
Live
RCAT: Slew Rates

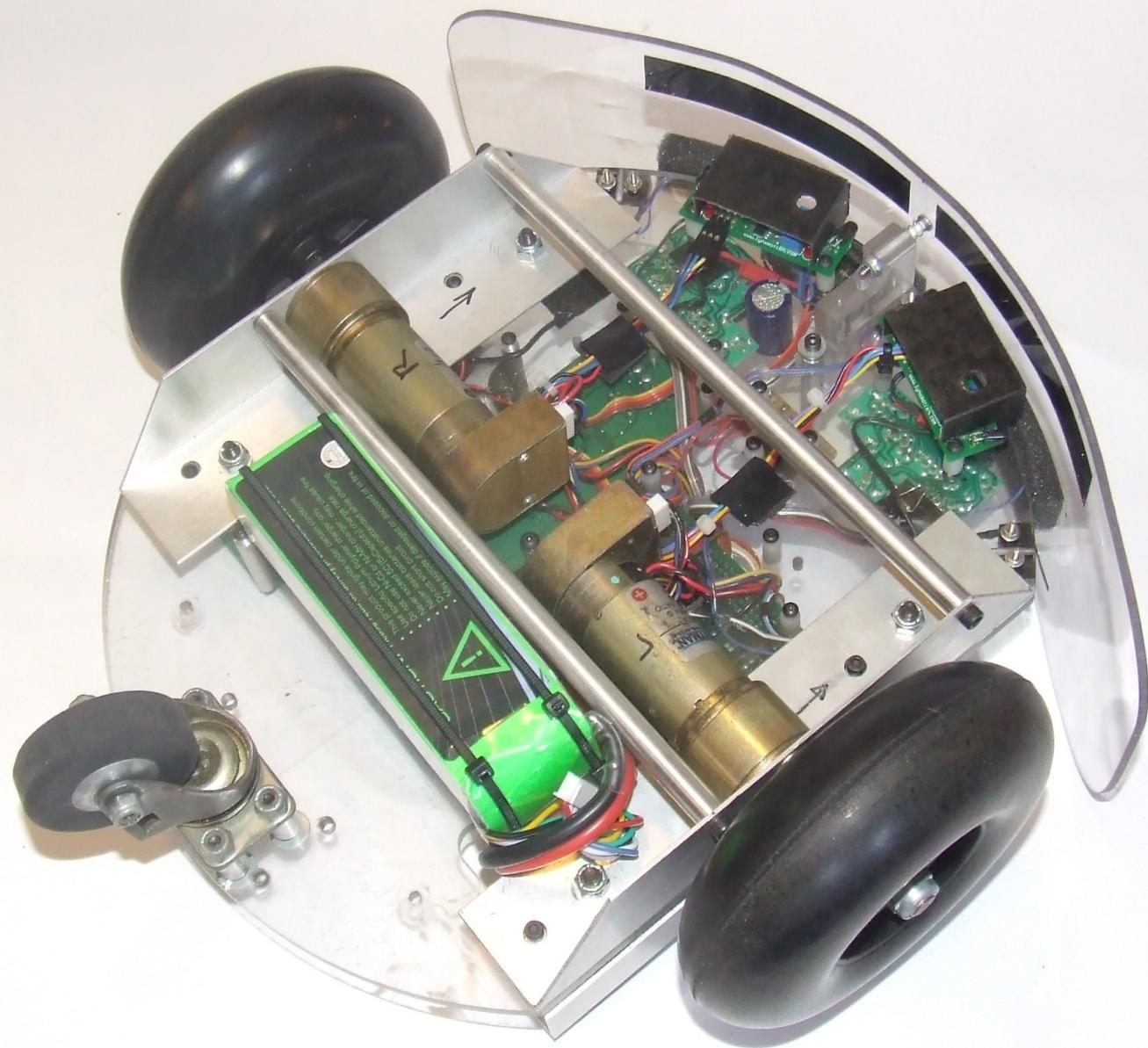
25 Hz Control Loop



25 Hz Control Loop





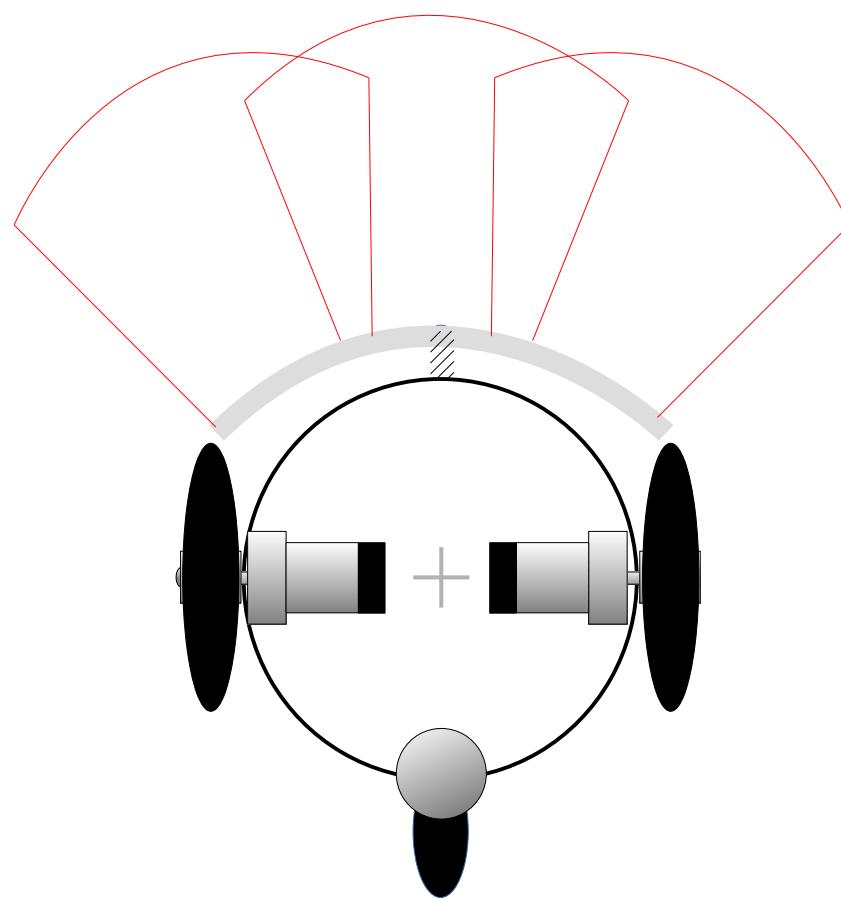


IR Proximity Detectors

Left

Center

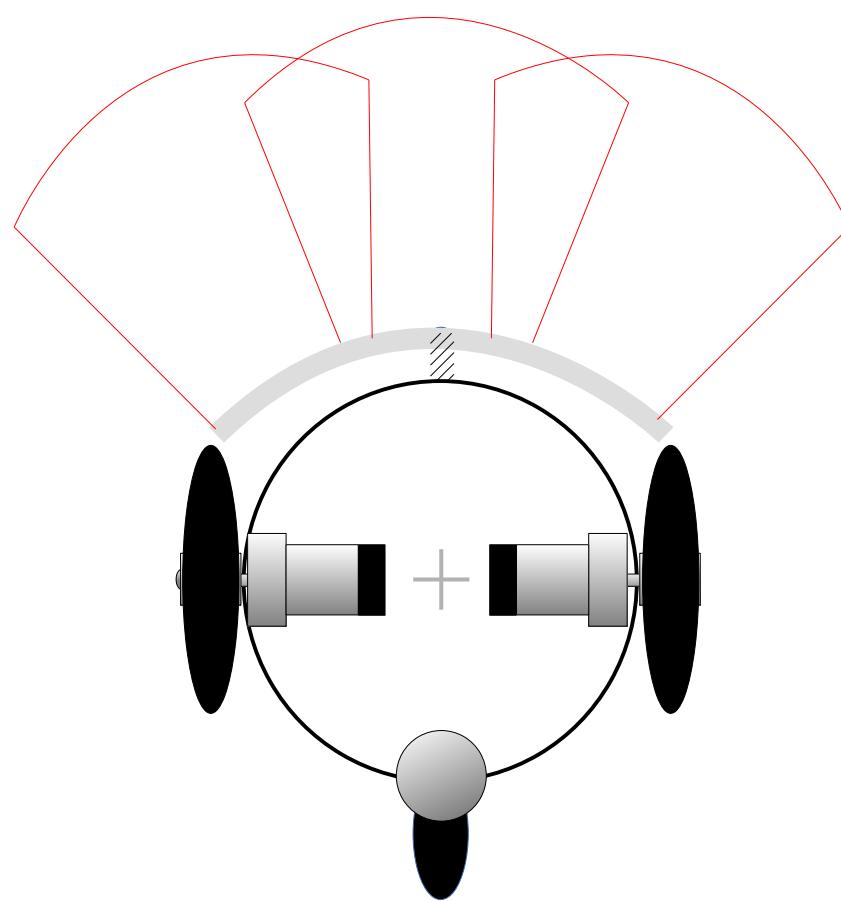
Right



IR Proximity Detectors

ir_read()

1 4 2
Left Center Right



IR Behavior

Call ir_read()

If NO detect, exit.

If LEFT detect, slow down, turn right.

If RIGHT detect, slow down, turn left.

If CENTER detect, slow to 0, keep turning...

```
// The PID controllers maintain left_velocity and right_velocity,  
// and also left_avg_velocity and right_avg_velocity (~1.25s moving window )  
  
int keep_turning ( int avg_flag )  
{  
    if ( avg_flag ) {      // use average velocity  
        if ( left_avg_velocity > right_avg_velocity ) return 1;  
        else return -1;  
    }                      // else use instantaneous velocity  
    if ( left_velocity > right_velocity ) return 1;  
    return -1  
}
```

```
CMDBLOCK ir;                                // output struct for arbiter()

void IR_Behavior( void )
{
    int detect = read_ir();                  // read IR sensors, 0 = NO DETECT,
                                              // 1 = LEFT, 2 = RIGHT, 4 = CENTER

    If ( detect == 0 )
        ir.flag = FALSE;                   // and exit
    else
    {   ir.flag = TRUE;
        if ( detect > 2 ) detect = CENTER;

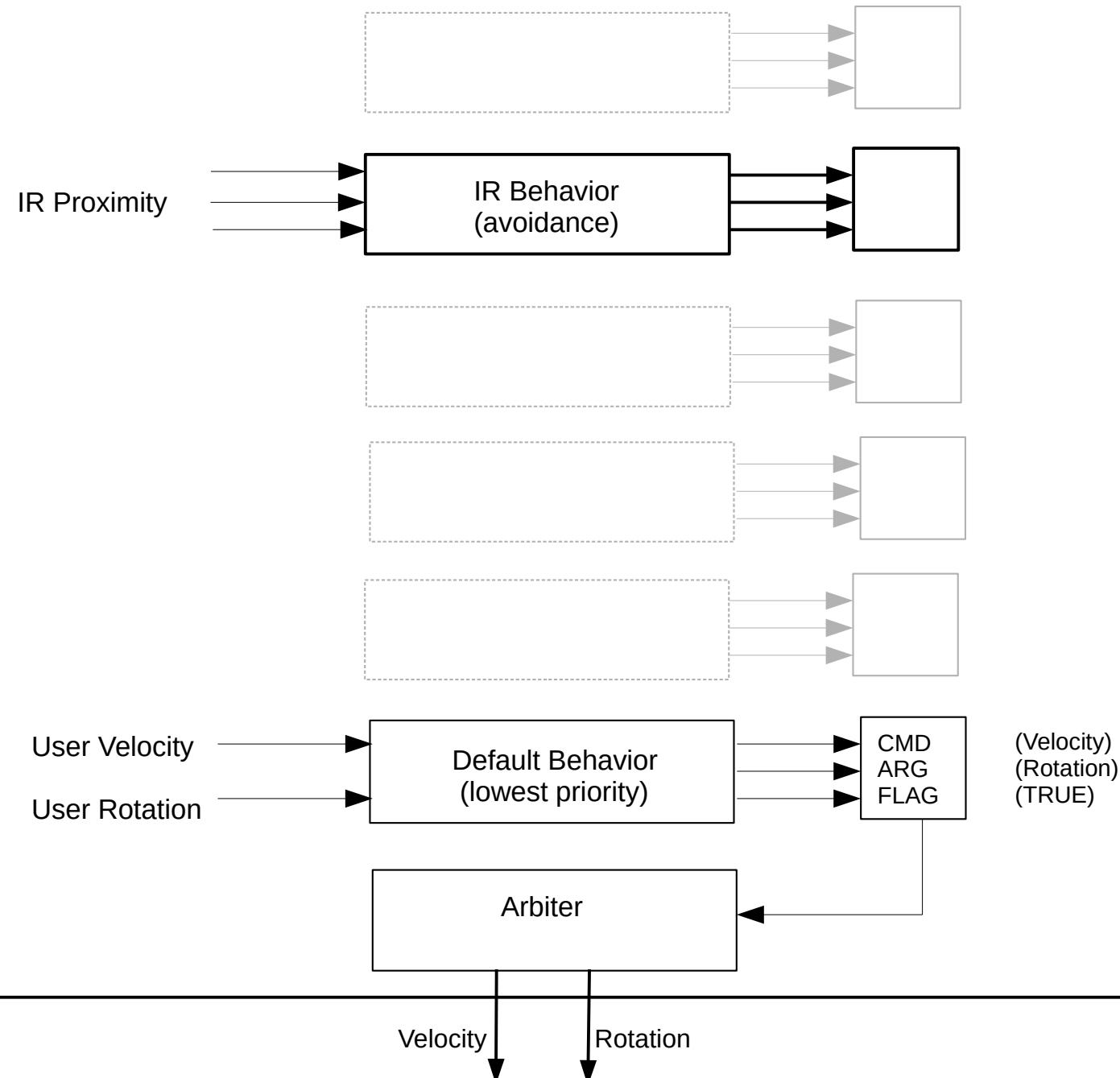
        switch ( detect ) {

            case LEFT : {                // left detect, slow to ½ speed
                // and turn right
                ir.cmd = slew_vel ( p_user_velocity/2, p_ir_vrate );
                ir.arg = slew_rot ( p_ir_turn, p_ir_rrate );
                break;
            }

            case RIGHT : {               // right detect, slow to ½ speed
                // and turn left
                ir.cmd = slew_vel ( p_user_velocity/2, p_ir_vrate );
                ir.arg = slew_rot ( -p_ir_turn, p_ir_rrate );
                break;
            }

            case CENTER : {             // center detect, slow to 0
                // and keep turning the "same" direction
                ir.cmd = slew_vel ( 0, p_ir_vrate );
                ir.arg = slew_rot ( p_ir_turn * keep_turning ( 0 ), p_ir_rrate );
                break;
            }
        }
    }
}
```

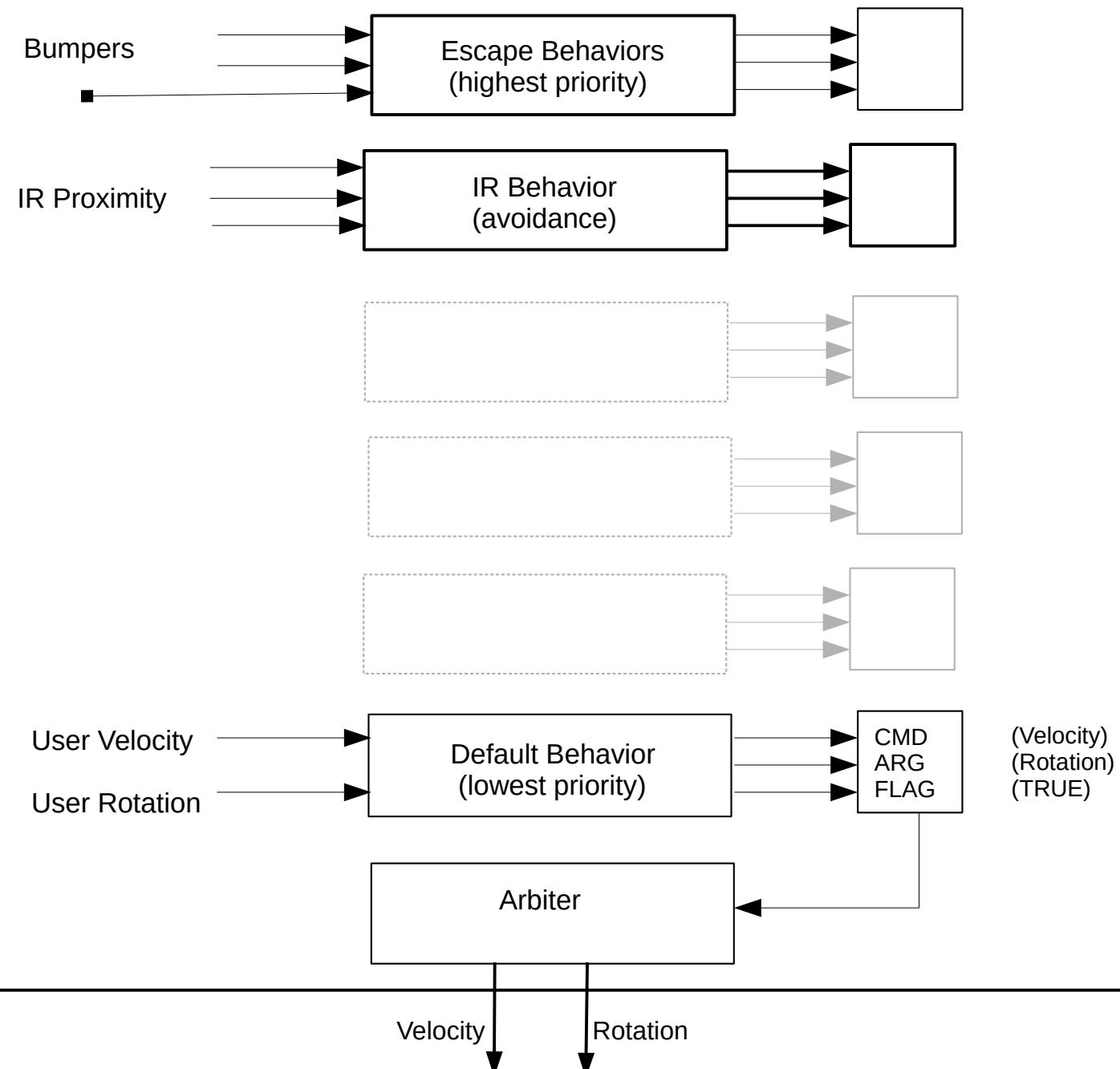
25 Hz Control Loop



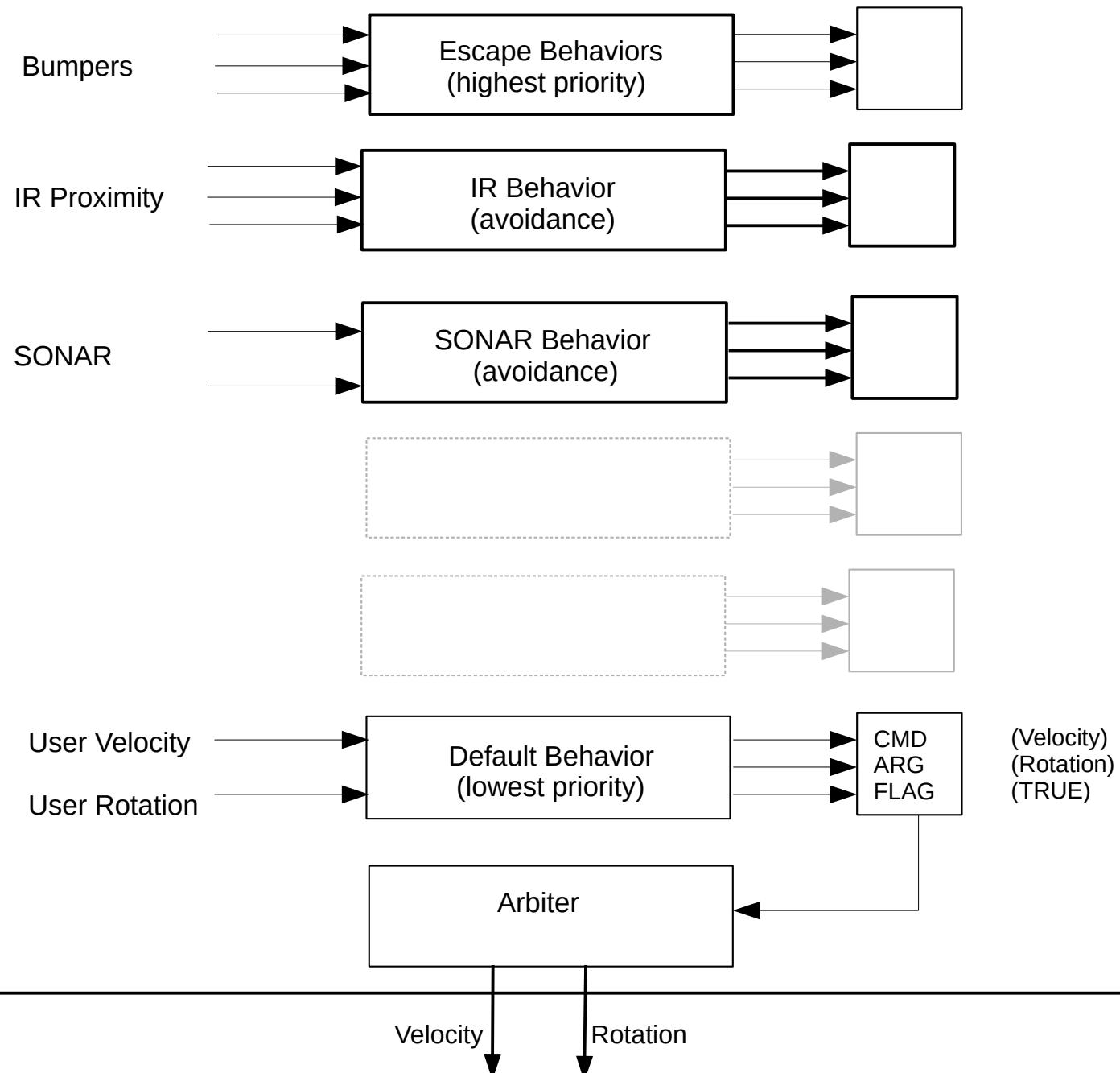
Demo 3

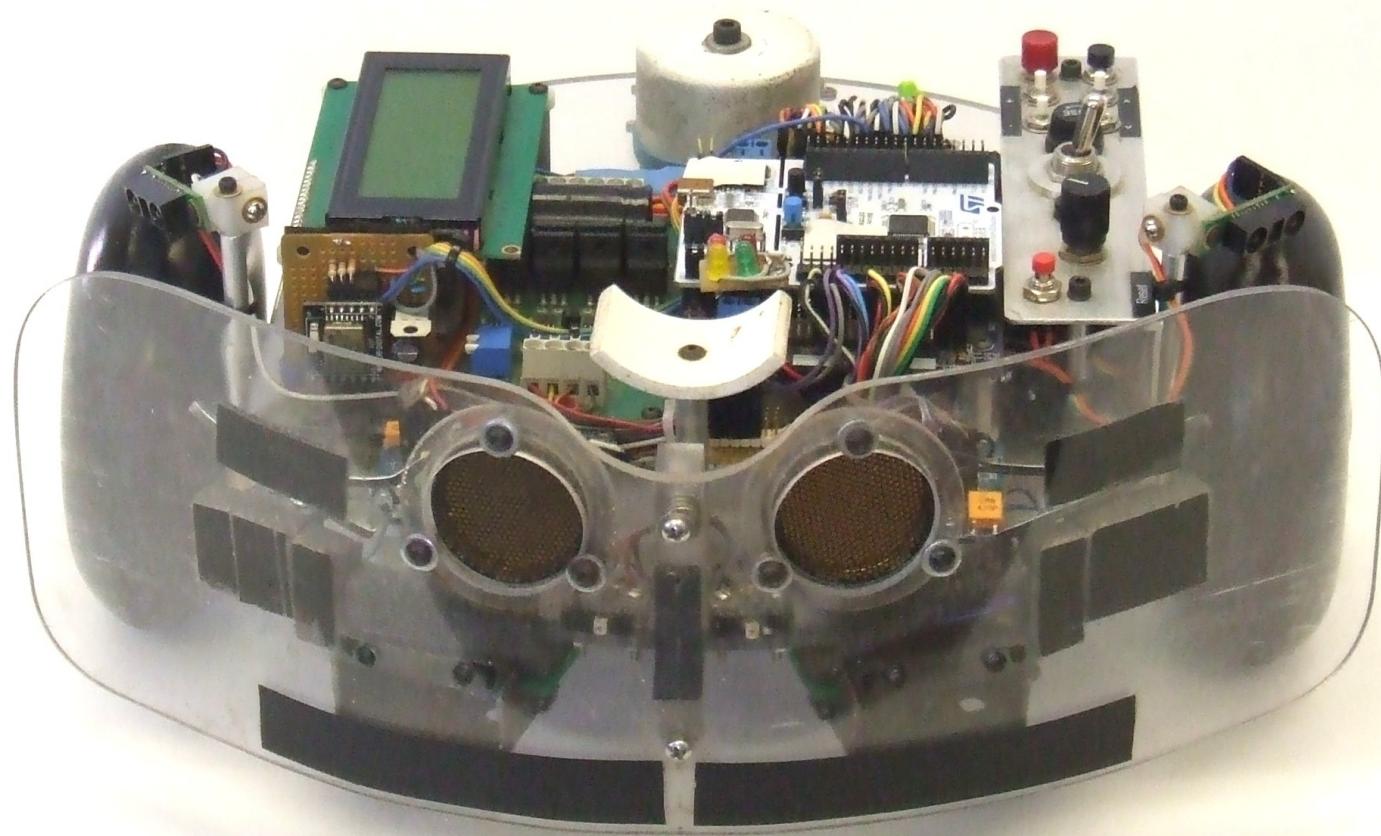
Live
RCAT: IR Avoidance

25 Hz Control Loop

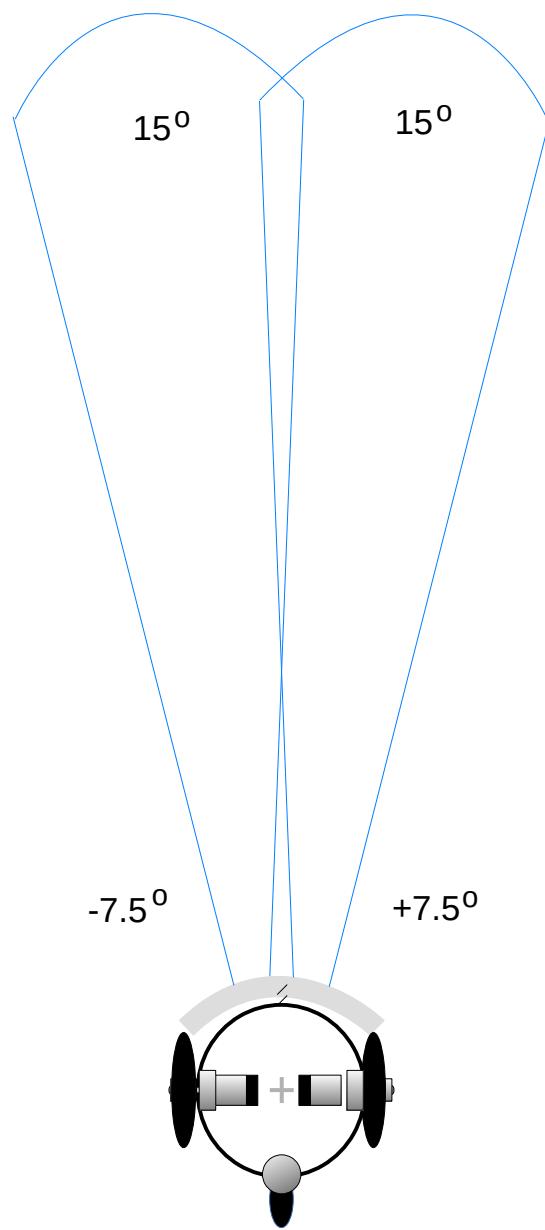


25 Hz Control Loop

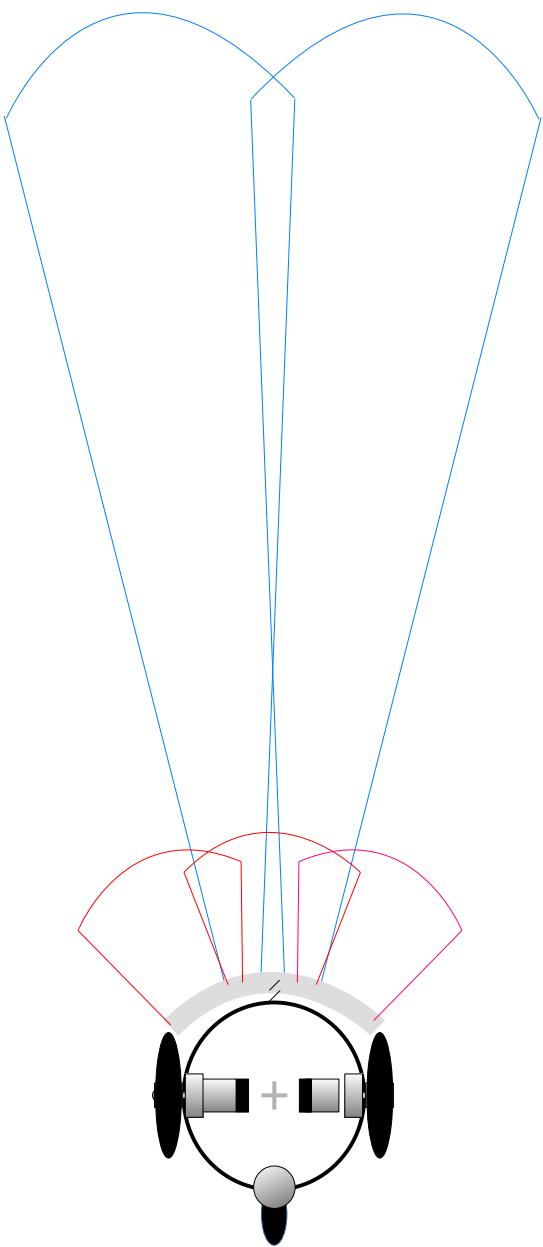




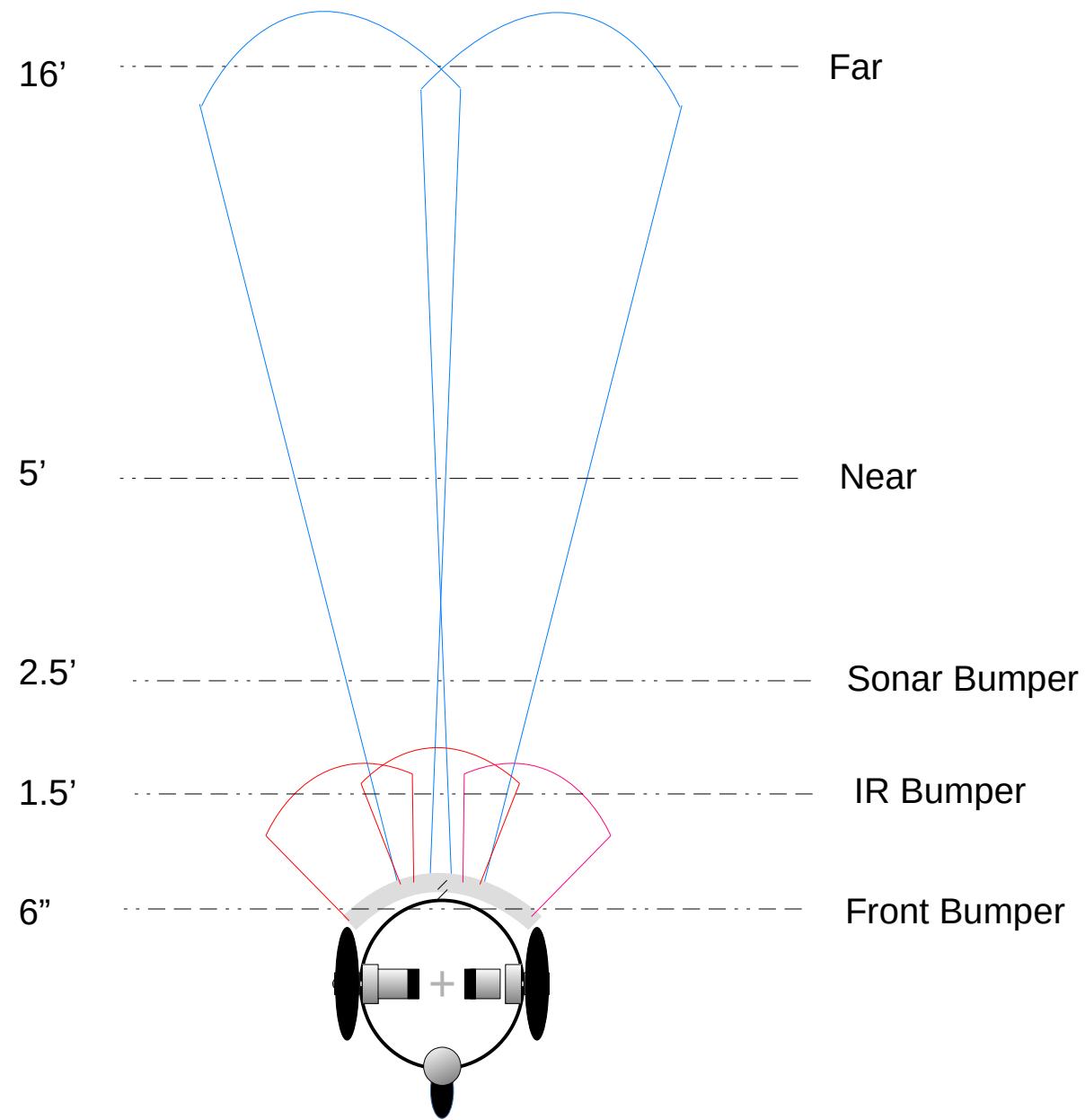
Stereo SONAR



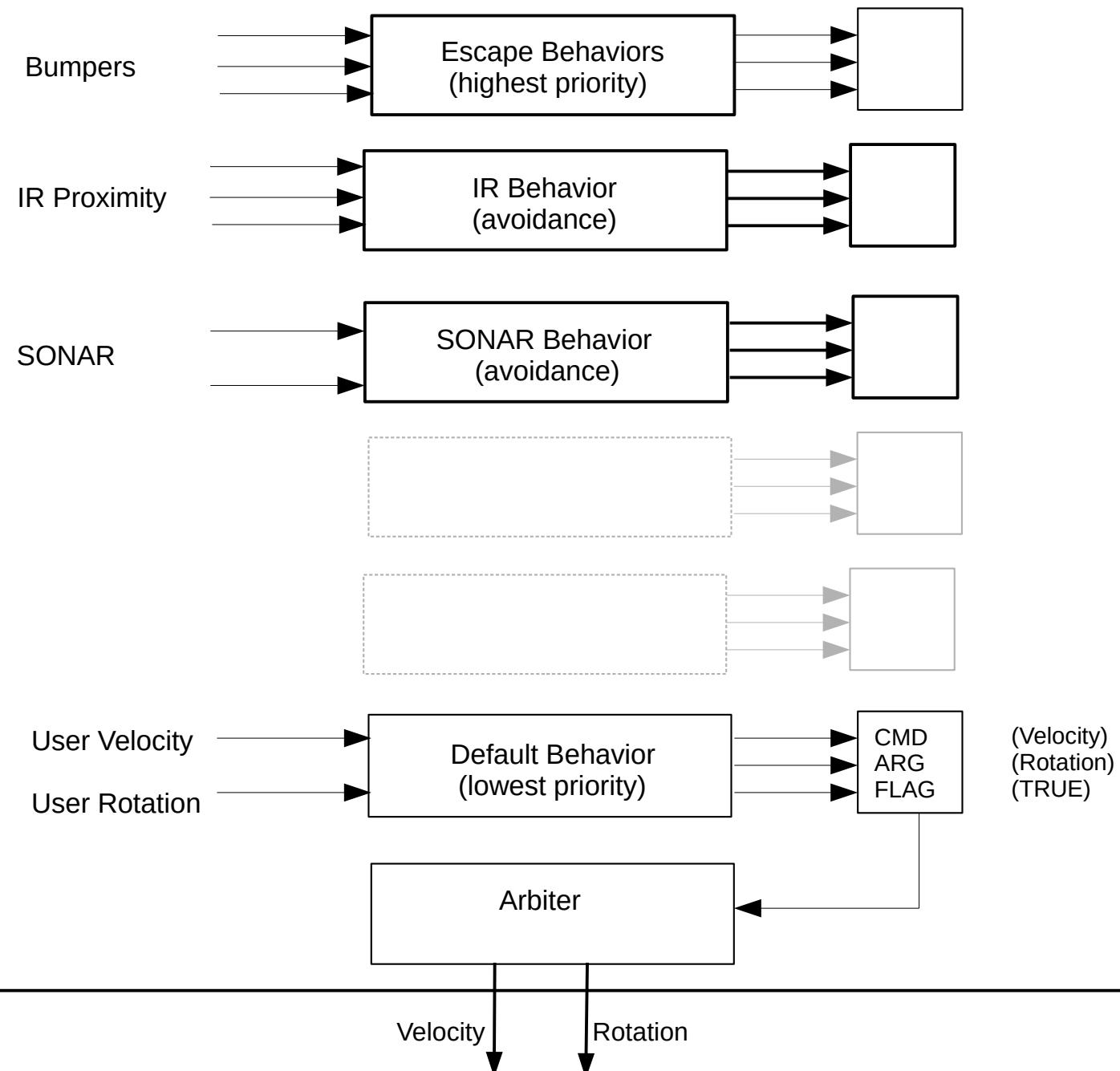
Stereo SONAR, IR Proximity



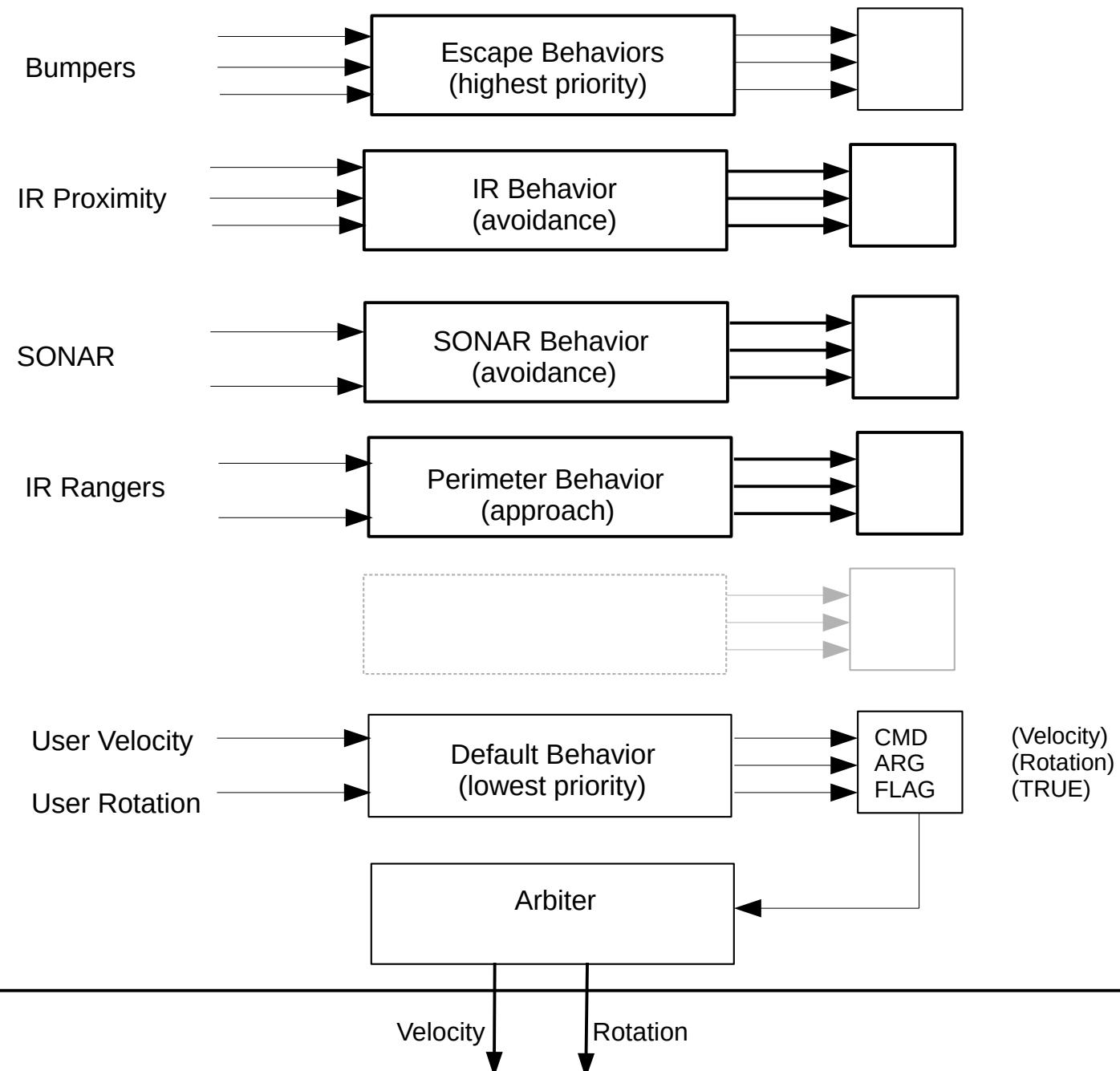
Stereo SONAR, IR Proximity

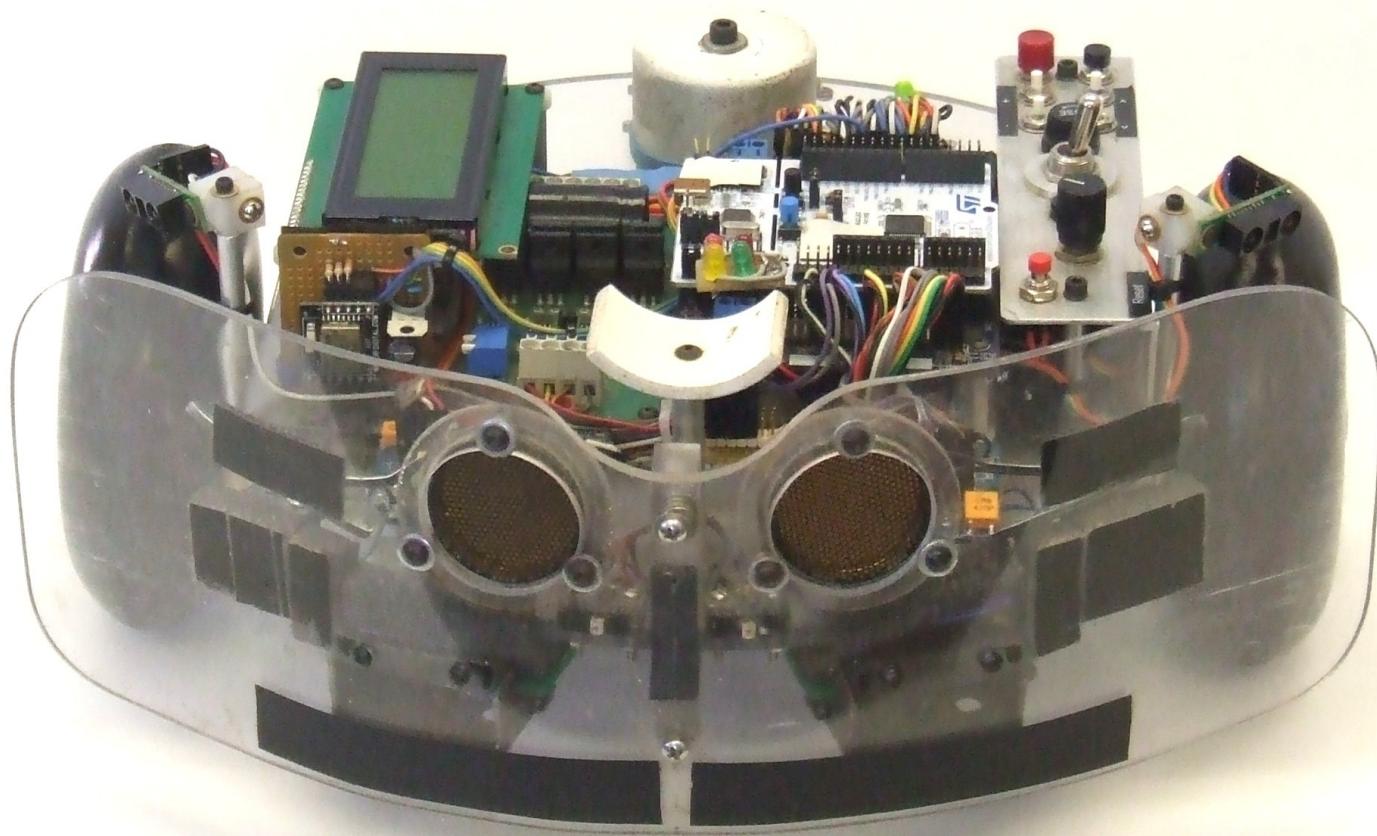


25 Hz Control Loop



25 Hz Control Loop

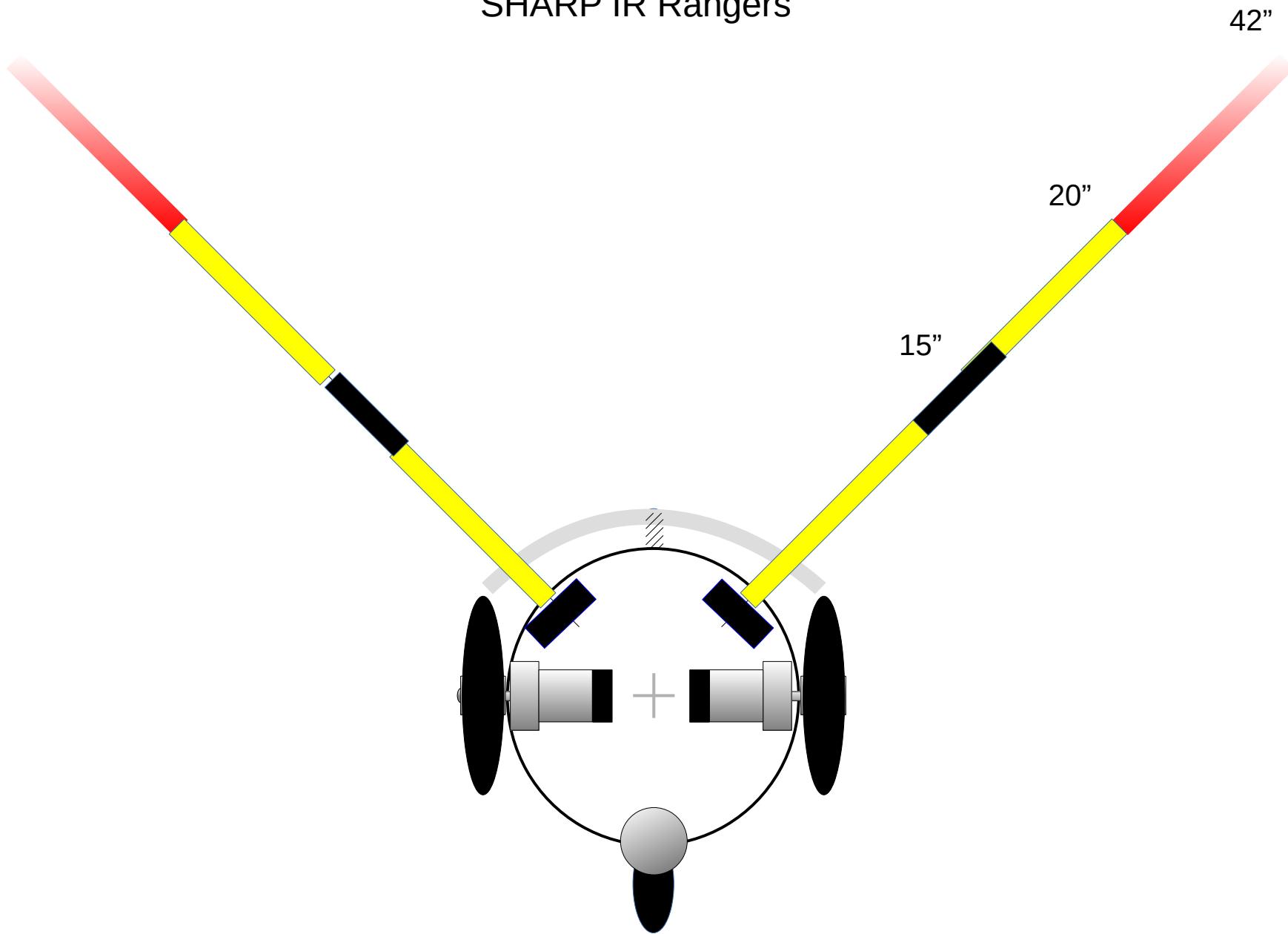




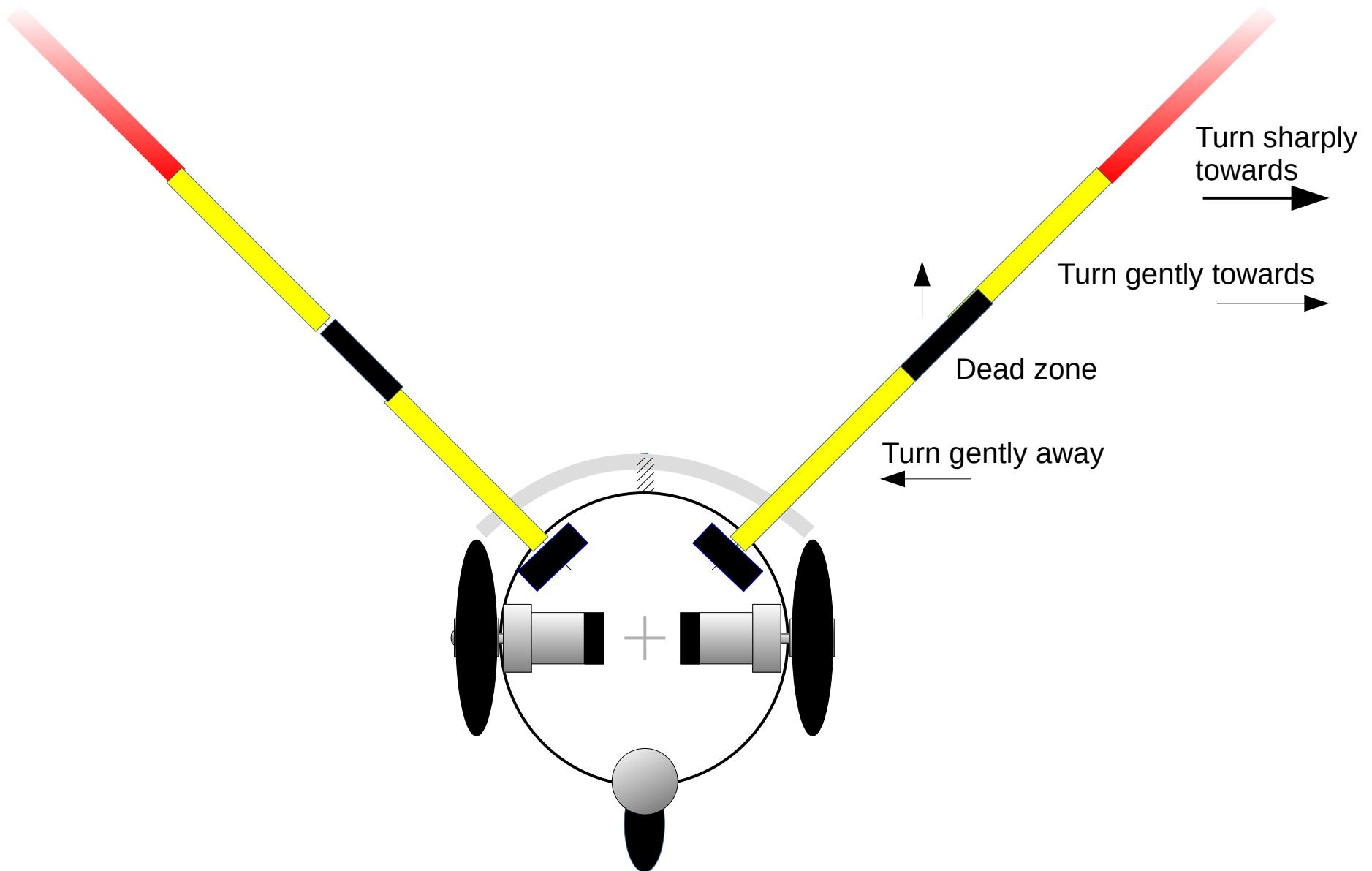
Perimeter Following / Maze Running
SHARP IR Rangers



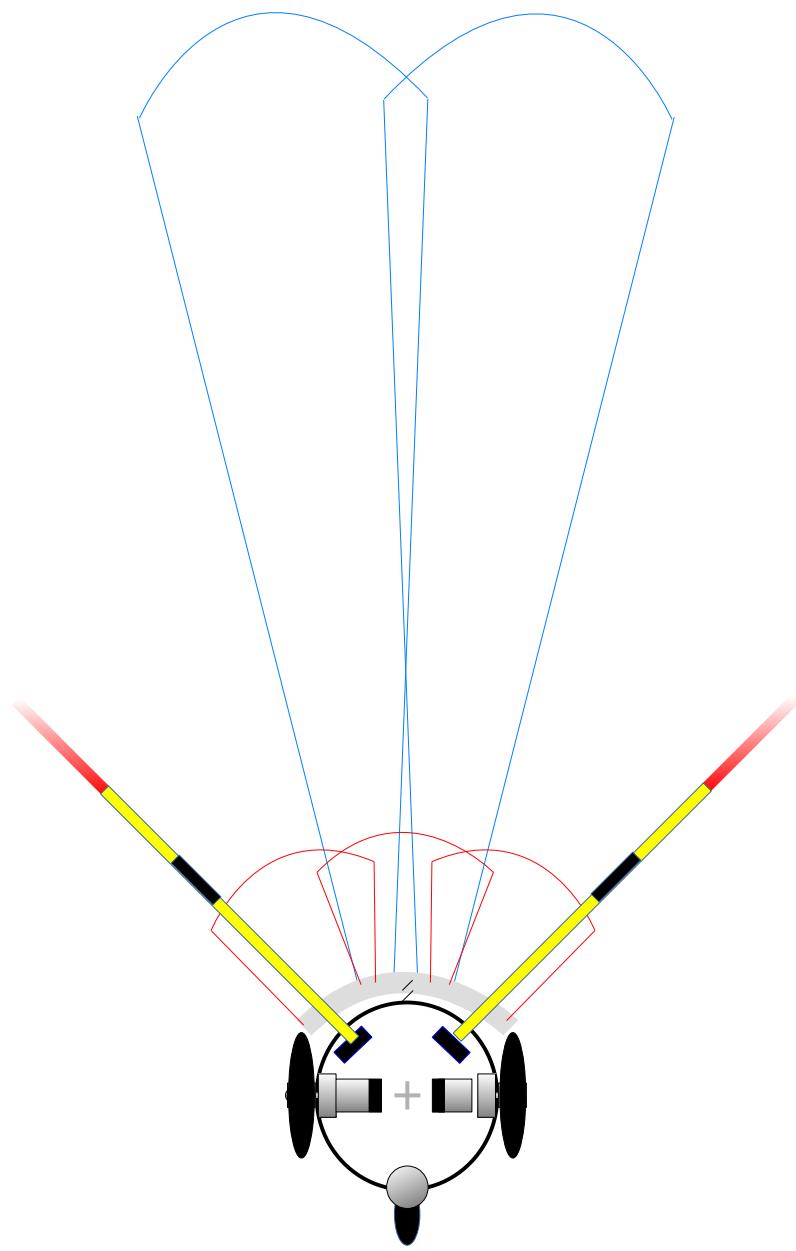
Perimeter Following / Maze Running
SHARP IR Rangers



Perimeter Following / Maze Running SHARP IR Rangers



Stereo SONAR, IR Proximity, IR Rangers

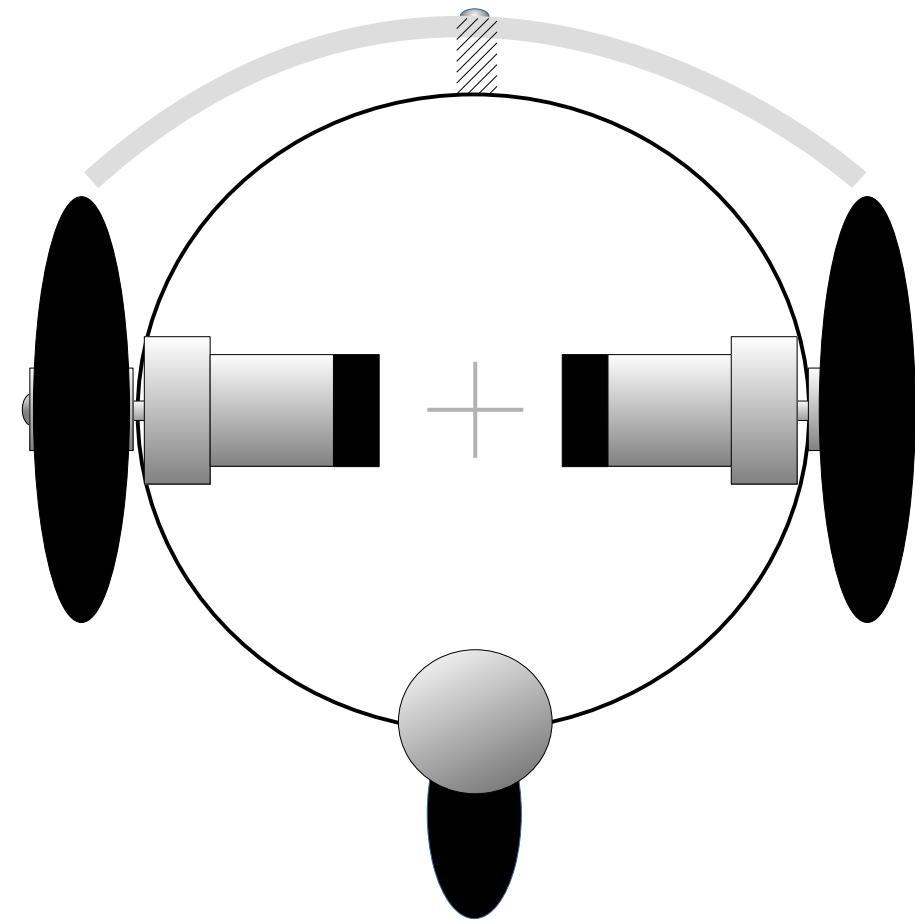


Demo 4

Videos:

RCAT: Perimeter Following
nBot: Perimeter Following

II. Waypoint Navigation



II. Waypoint Navigation

- A. Determining Pose
- B. A Navigation Virtual Sensor
- C. A Navigation Behavior

A. Determining POSE

0

+Y

0

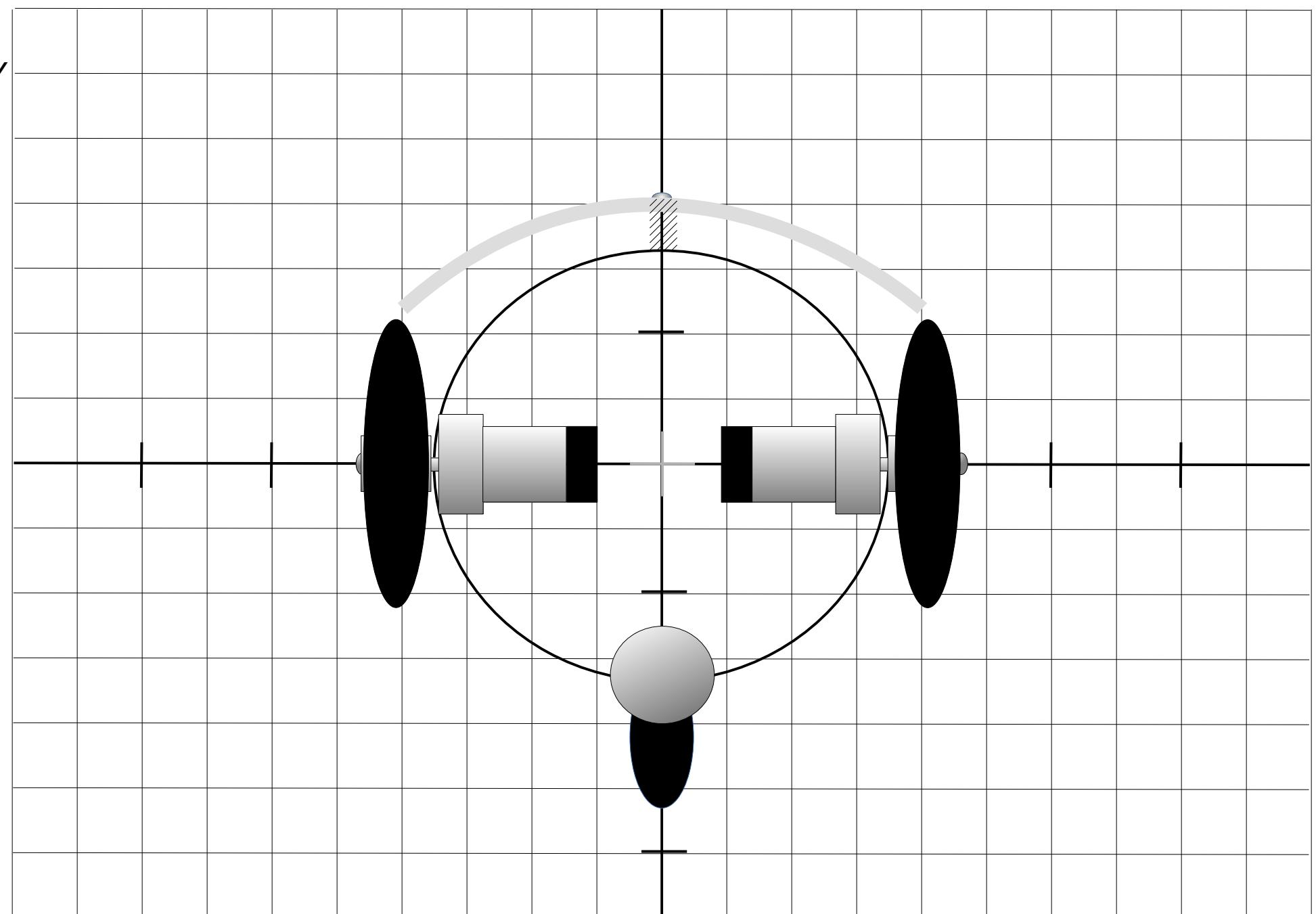
0

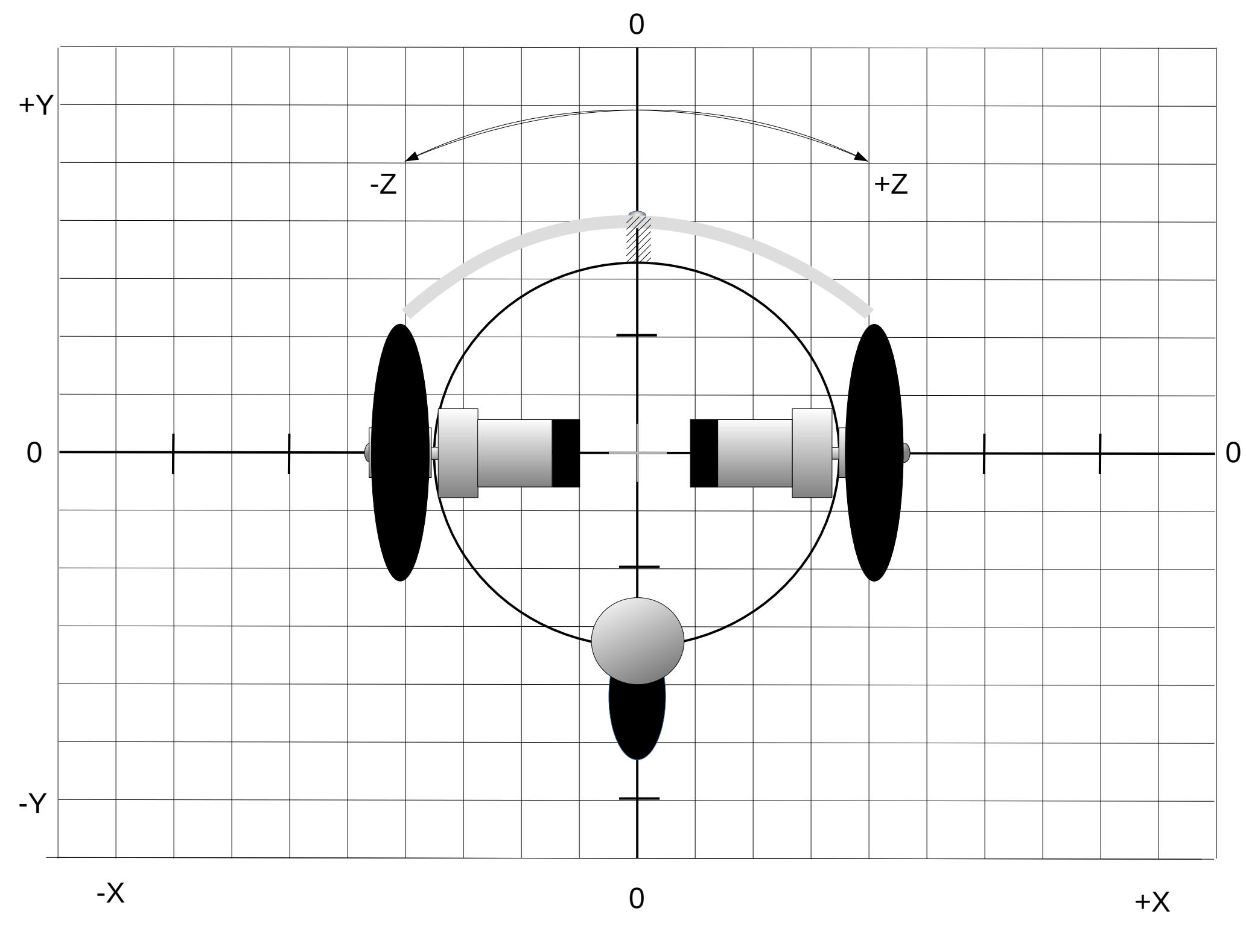
-Y

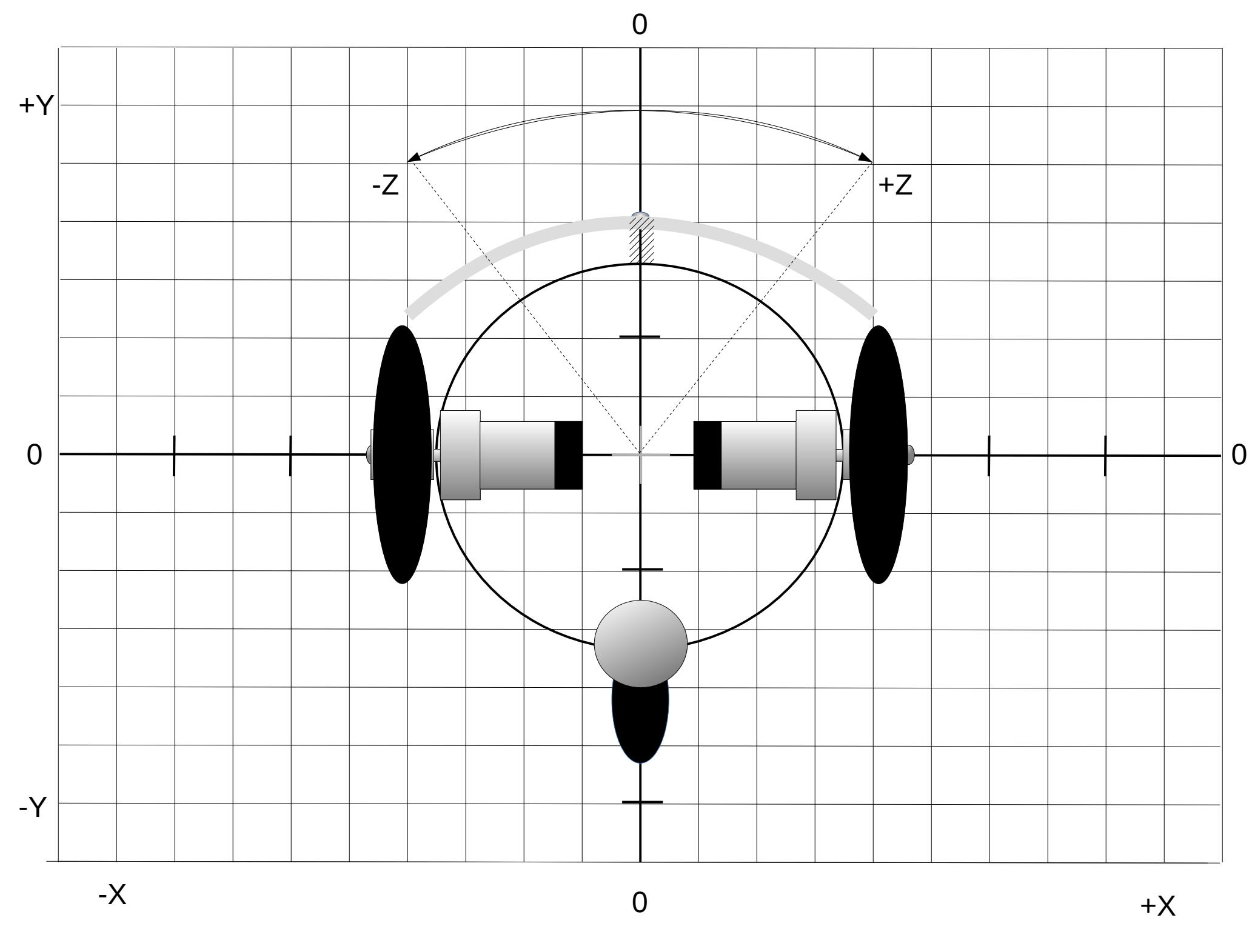
-X

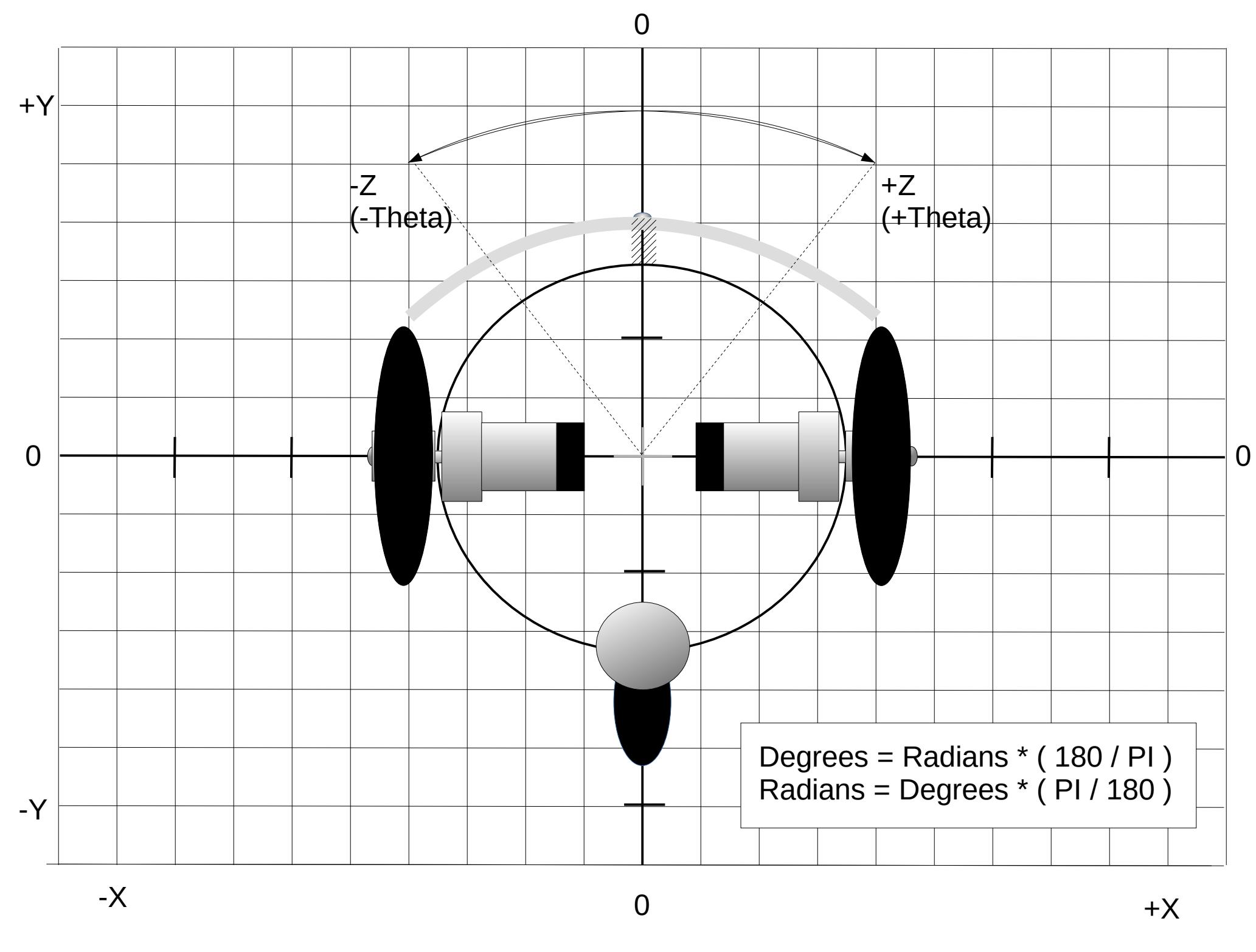
0

+X

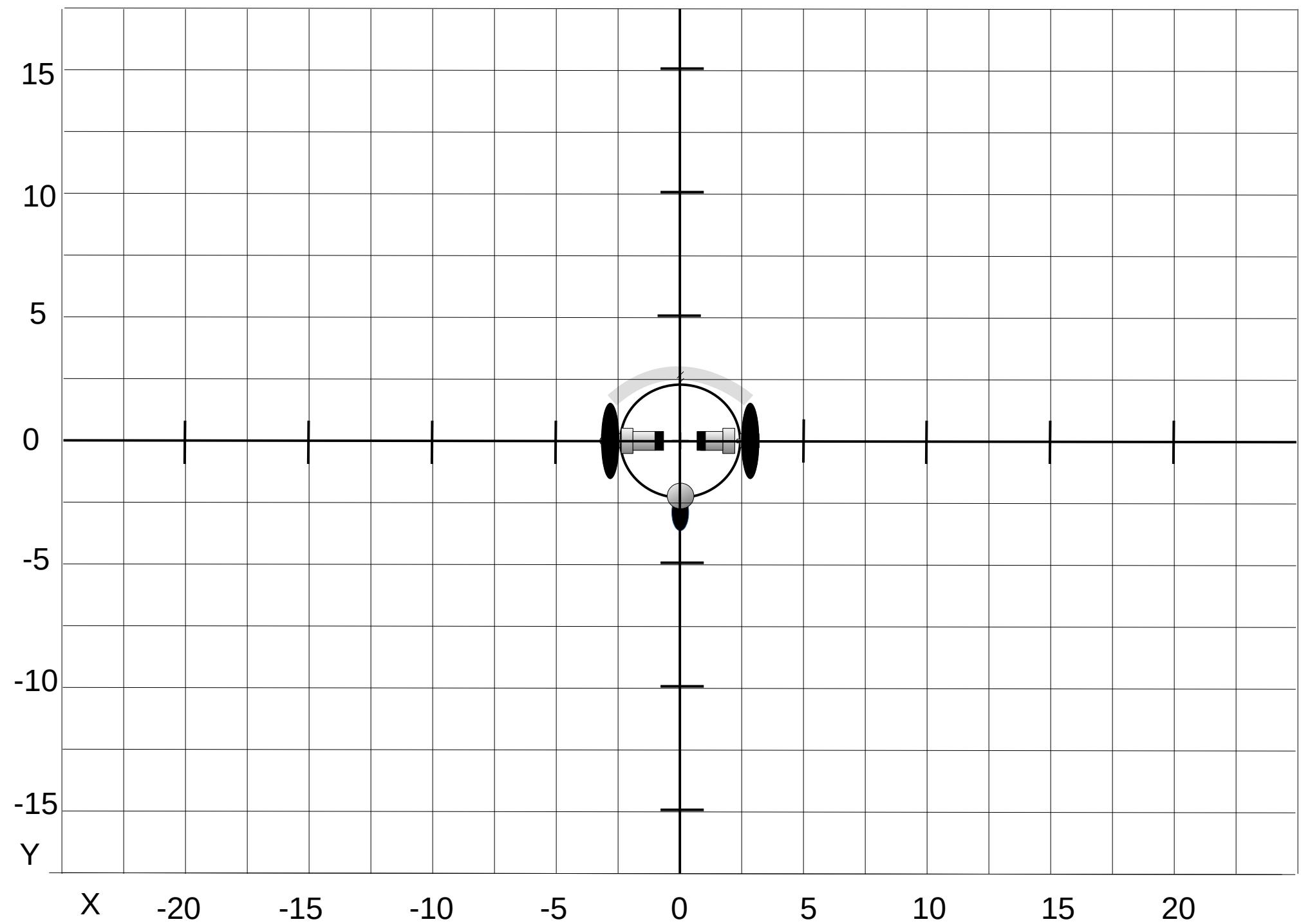




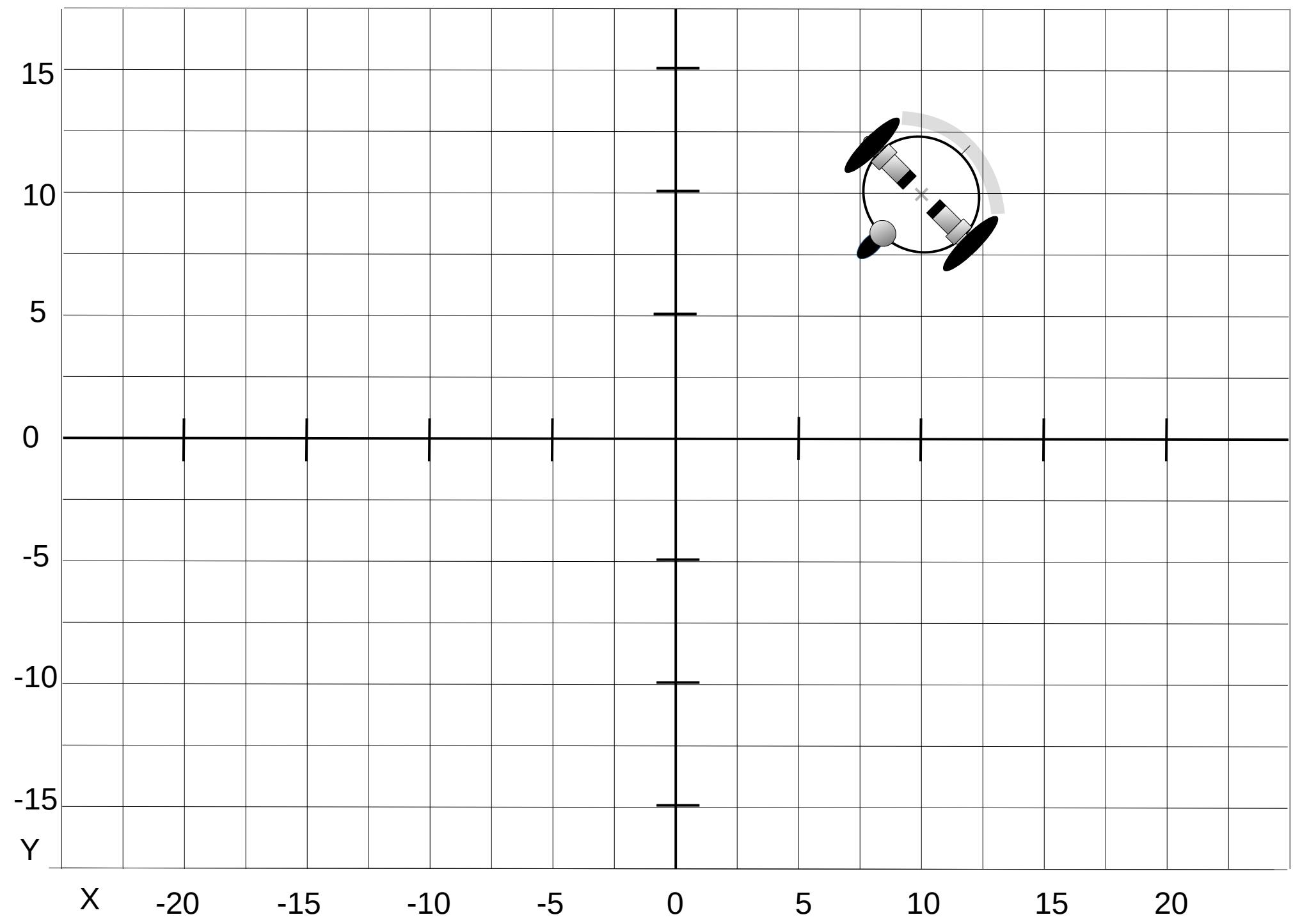




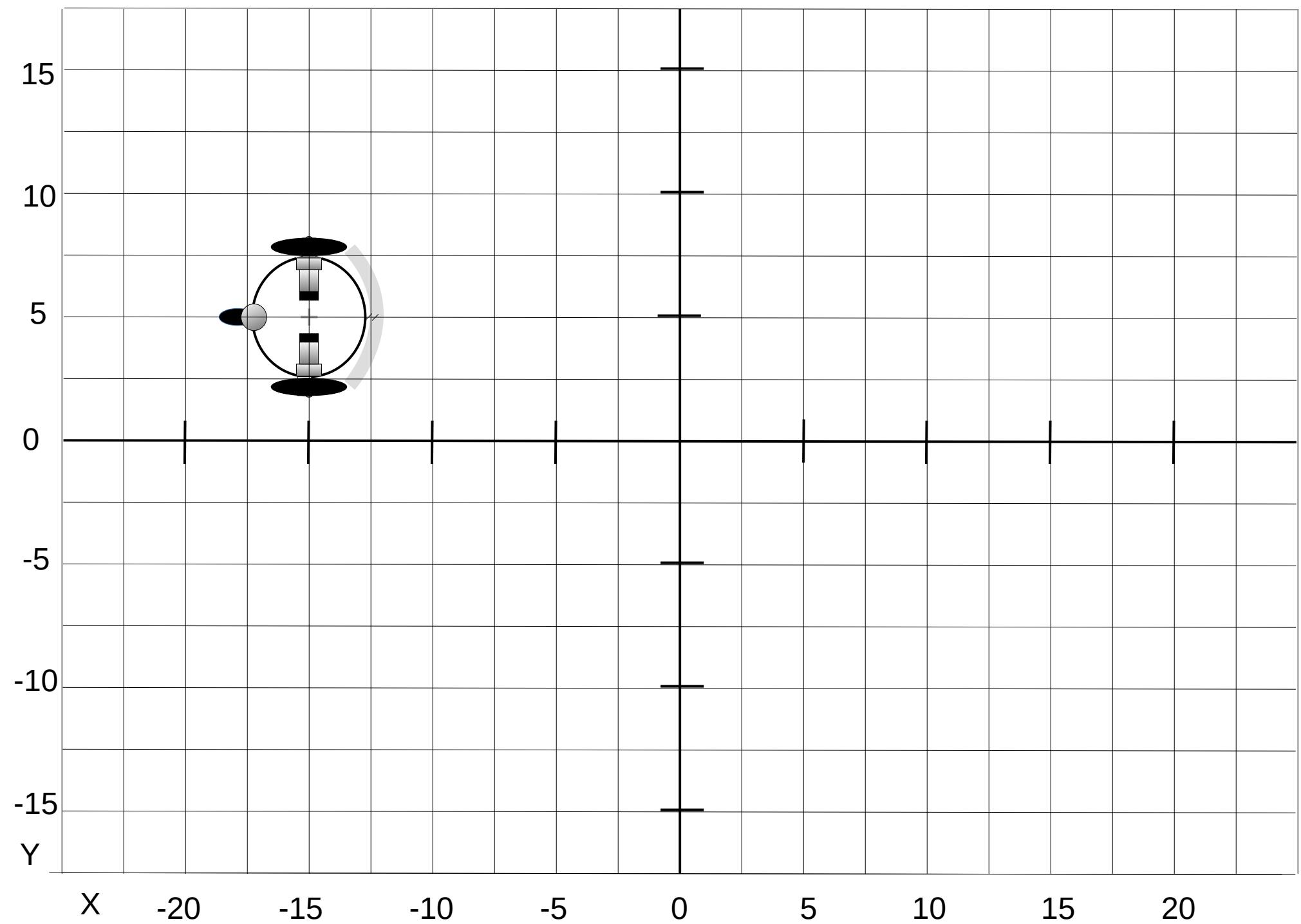
Pose: X = 0, Y = 0, Z = 0



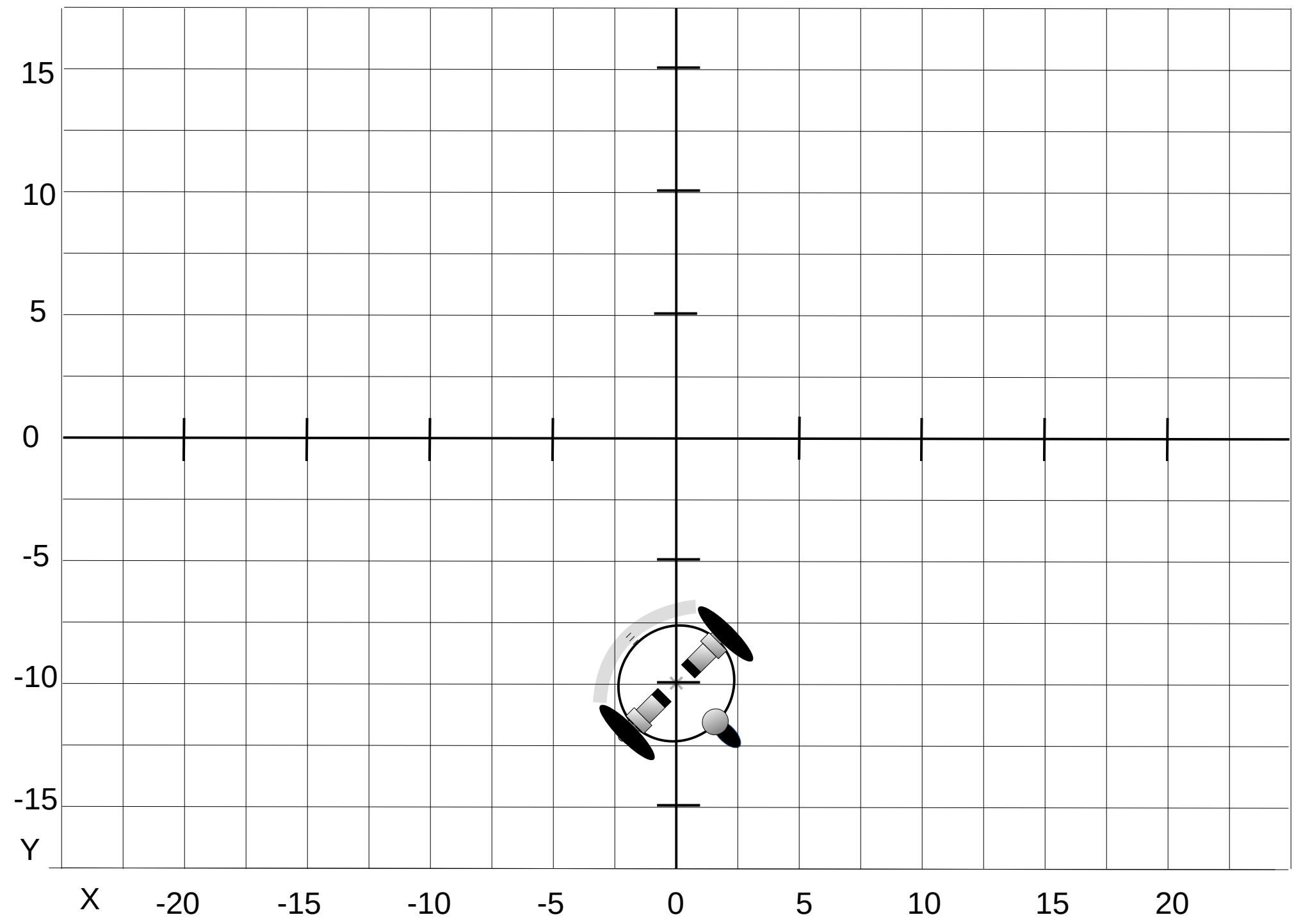
Pose: X = 10, Y = 10, Z = 45



Pose: X = -15, Y = 5, Z = 90



Pose: X = 0, Y = -10, Z = -45



Determining POSE:

```
float X_position;           // x coordinate in inches  
float Y_position;          // y coordinate in inches  
float Theta;               // z coordinate in radians
```

Multiple methods exists for determining POSE: radio beacons, RTK GPS, RealSense T265 imu+depth sense camera, optical flow, and so forth.

The method of navigation described in this talk does not depend on how POSE is determined. However, the robots in these demonstrations and videos all use some form of wheel odometry to determine POSE.

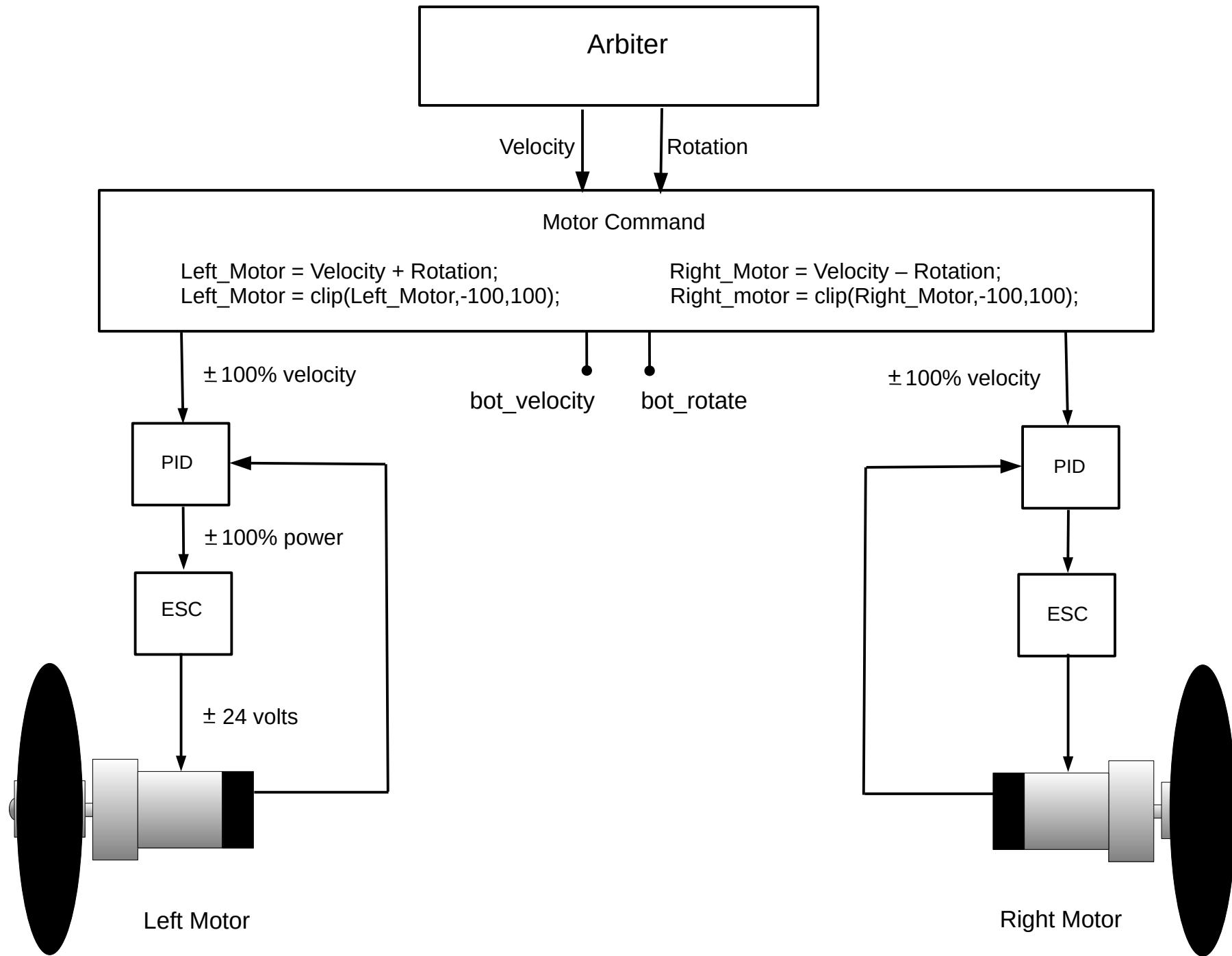
Fun Facts to Know and Tell:

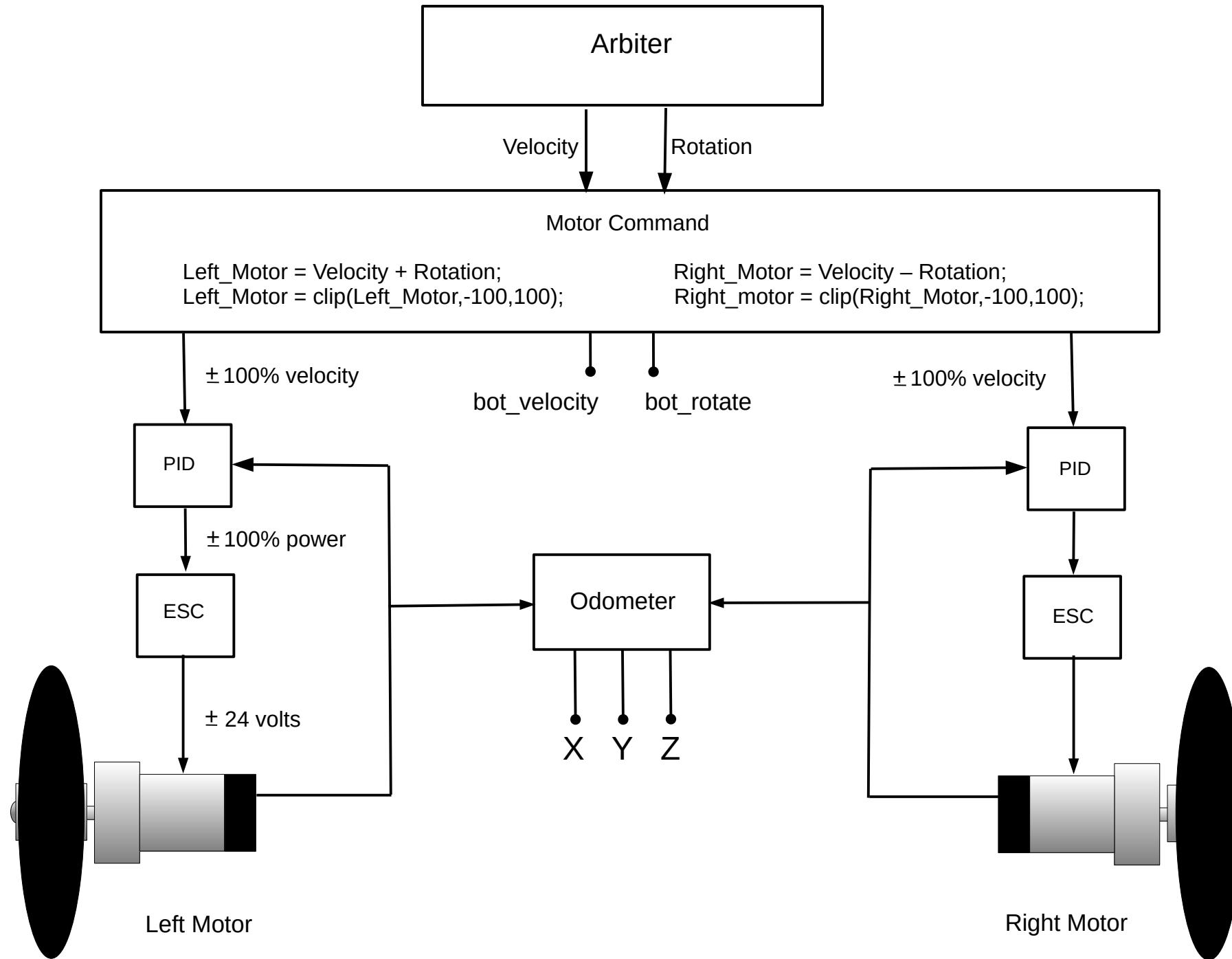
Odometry: Greek hodometron, from hodos way, road + metron measure.

Archimedes' hodometron was used by Romans to layout mile markers.

Three forms of odometry used on my robots:

1. Wheel Odometry (nBot, LegoBot)
2. Wheel Odometry + Gyro (Rcat)
3. Wheel Odometry + IMU (jBot)





Determining POSE

1. Wheel Odometry

```
float X_position;          // persistent variable X in inches
float Y_position;          // persistent variable Y in inches
float Theta;                // persistent variable Z in radians

void odometer ( void )
{
    left_inches = read_left_encoder() * LEFT_CLICKS_PER_INCH;
    right_inches = read_right_encoder() * RIGHT_CLICKS_PER_INCH;

    inches = ( left_inches + right_inches ) / 2.0;
    wheel_rate = ( left_inches - right_inches ) / WHEEL_BASE;

    Theta += wheel_rate;
    Theta -= (float) ( (int) ( Theta / TWOPI ) ) * TWOPI;    // clip to +- 360

    X_position += inches * sin ( Theta );
    Y_position += inches * cos ( Theta );
}
```

Determining POSE

2. Wheel Odometry + Rate Gyro

```
void odometer ( void )
{
    left_inches = read_left_encoder() * LEFT_CLICKS_PER_INCH;
    right_inches = read_right_encoder() * RIGHT_CLICKS_PER_INCH;

    inches = ( left_inches + right_inches ) / 2.0;
    wheel_rate = ( left_inches - right_inches ) / WHEEL_BASE;

    gyro_rate = read_rate_gyro();
    gyro_rate = ( gyro_rate * ( PI / 180 ) ) / 25;

    if ( p_gyro_enable )
        Theta += gyro_rate;
    else
        Theta += wheel_rate;

    Theta -= (float) ( (int) ( Theta / TWOPI ) ) * TWOPI;

    X_position += inches * sin ( Theta );
    Y_position += inches * cos ( Theta );
}
```

Determining POSE

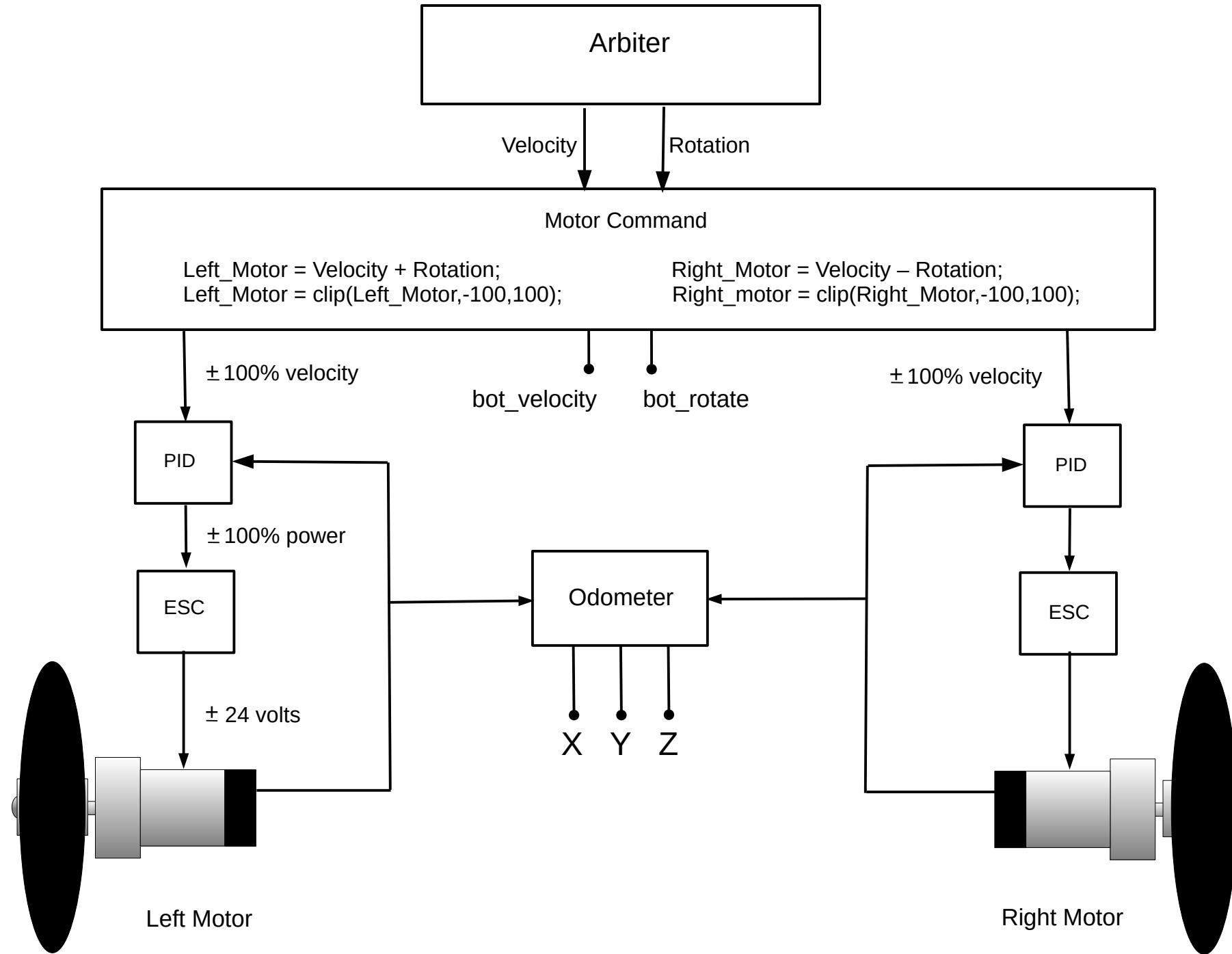
3. Wheel Odometry + IMU (Euler angles)

```
void odometer ( void )
{
    left_inches = read_left_encoder() * LEFT_CLICKS_PER_INCH;
    right_inches = read_right_encoder() * RIGHT_CLICKS_PER_INCH;

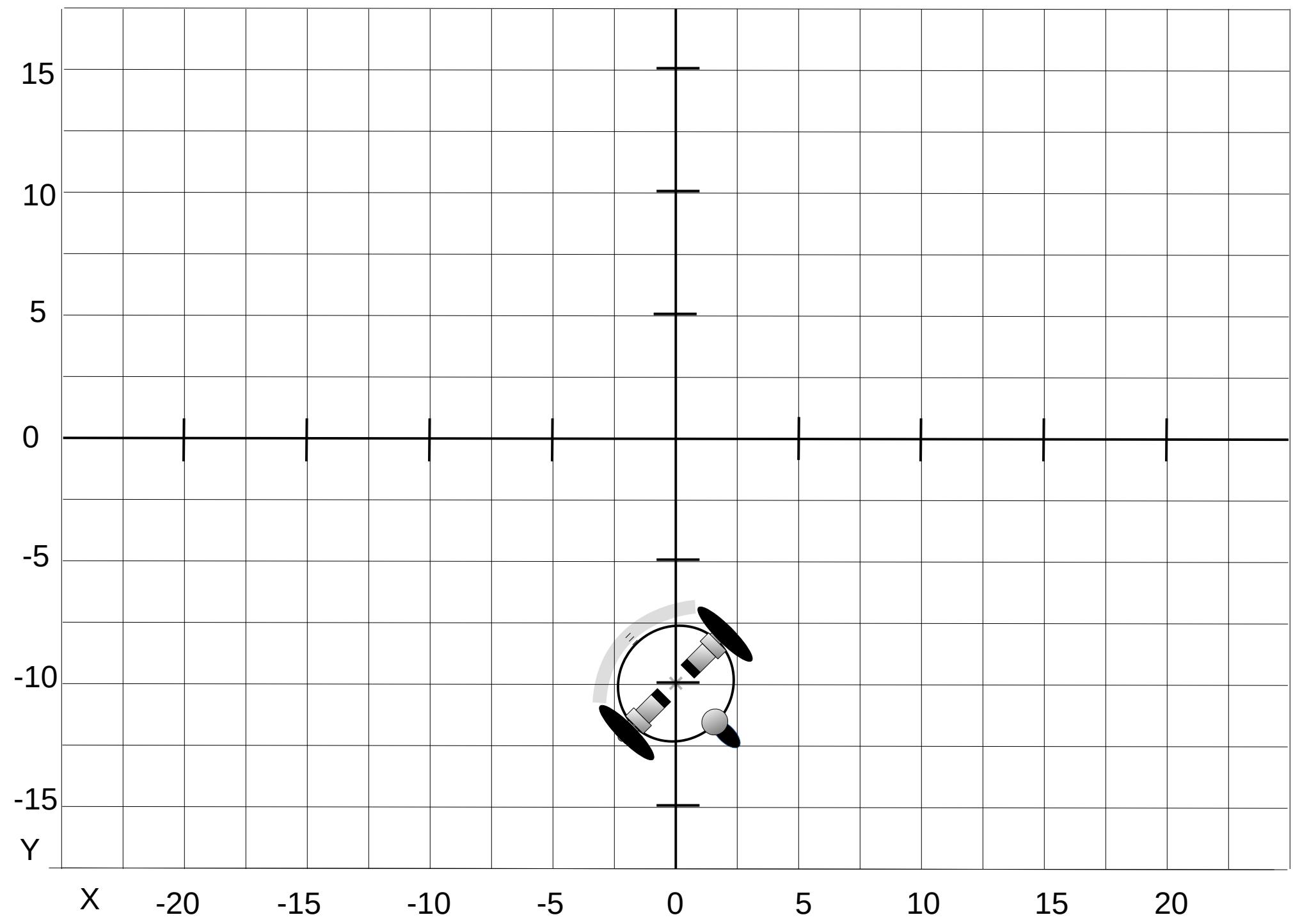
    inches = ( left_inches + right_inches ) / 2.0;
    wheel_rate = ( left_inches - right_inches ) / WHEEL_BASE;

    if (p_imu_enable) Theta = ( imu.yaw * ( PI / 180 ) ) + north_offset;
    else {
        Theta += wheel_rate;
        Theta -= (float) ( (int) ( Theta / TWOPI ) ) * TWOPI;
    }

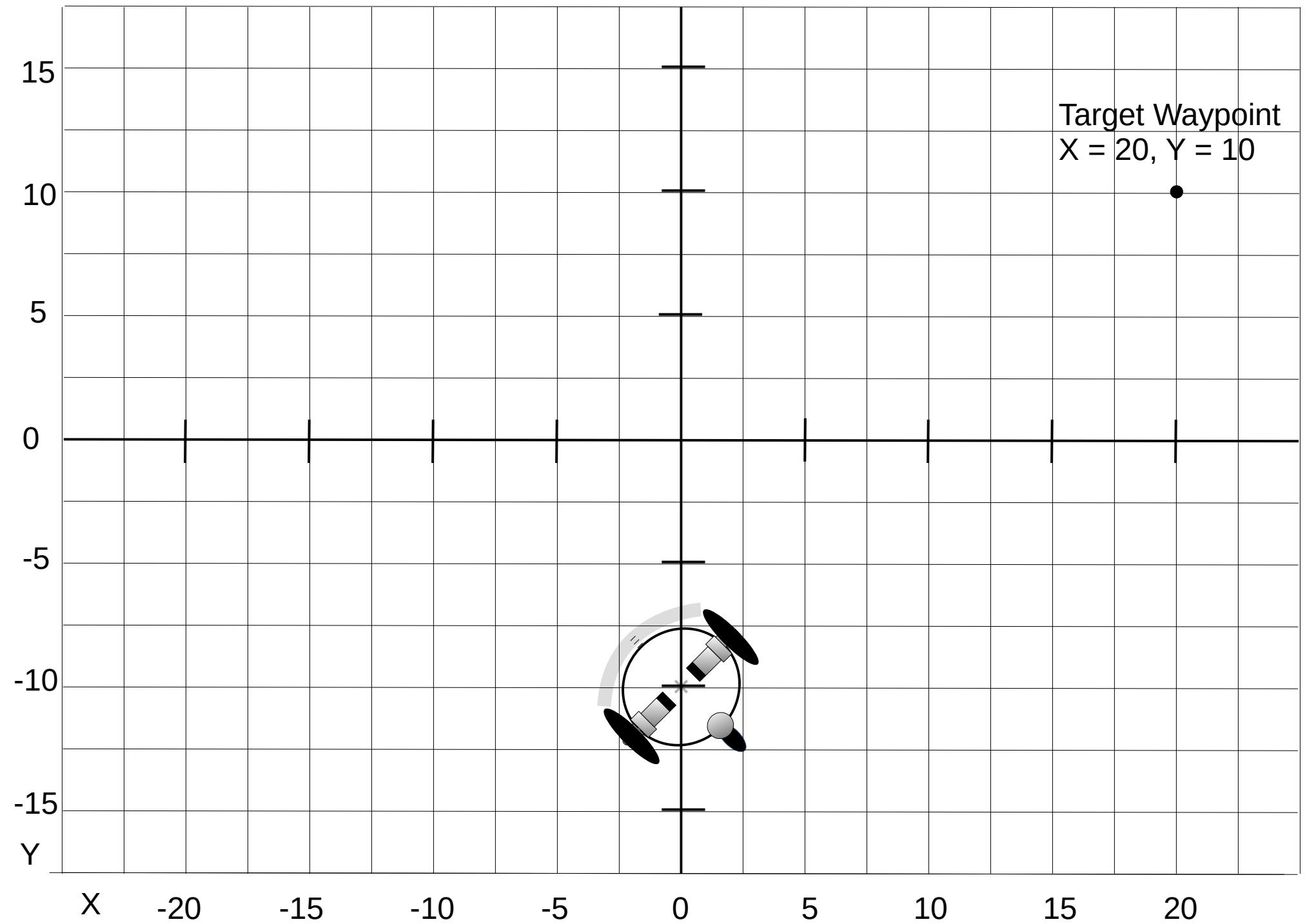
    X_position += inches * sin ( Theta );
    Y_position += inches * cos ( Theta );
}
```



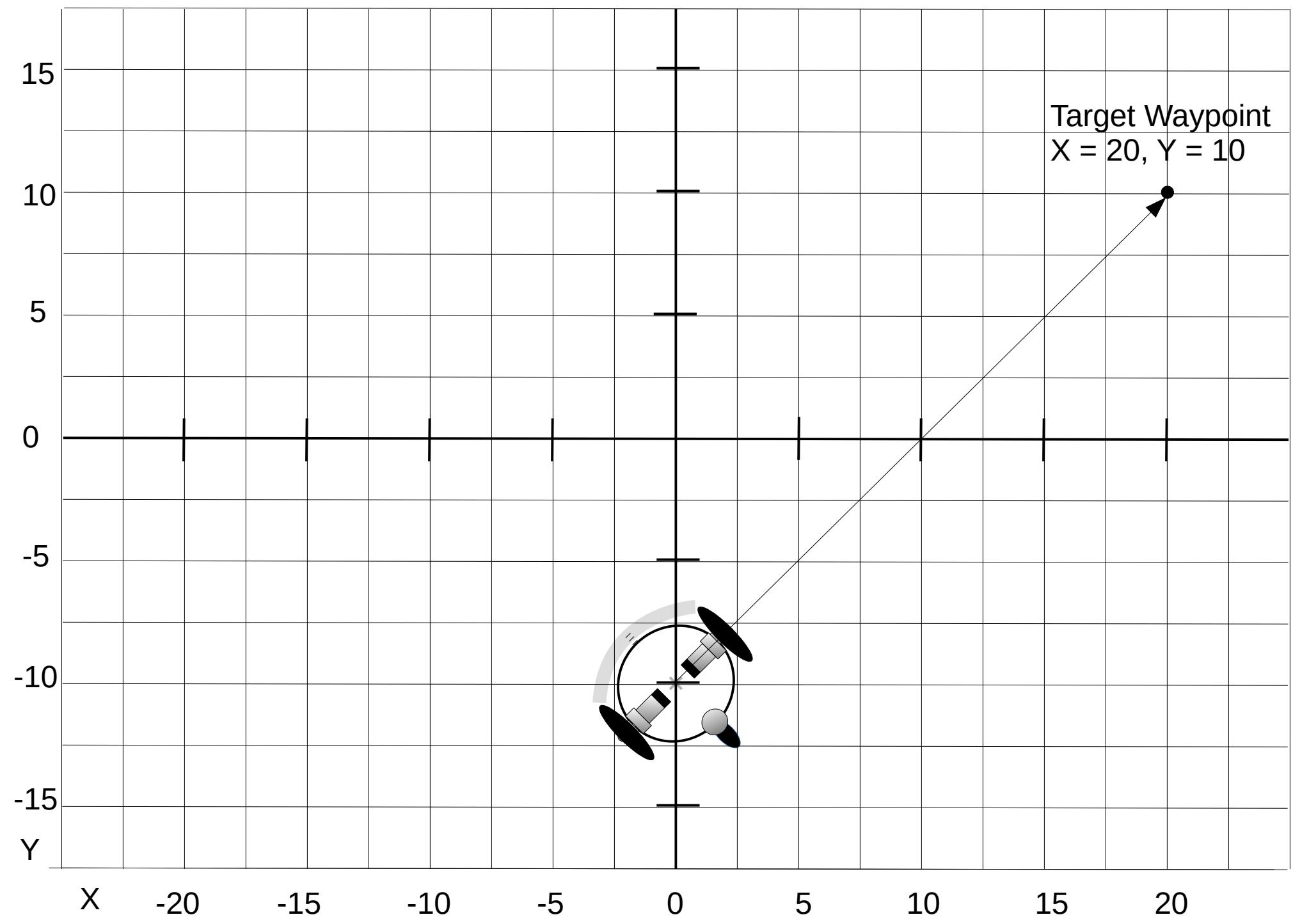
Pose: X = 0, Y = -10, Z = -45



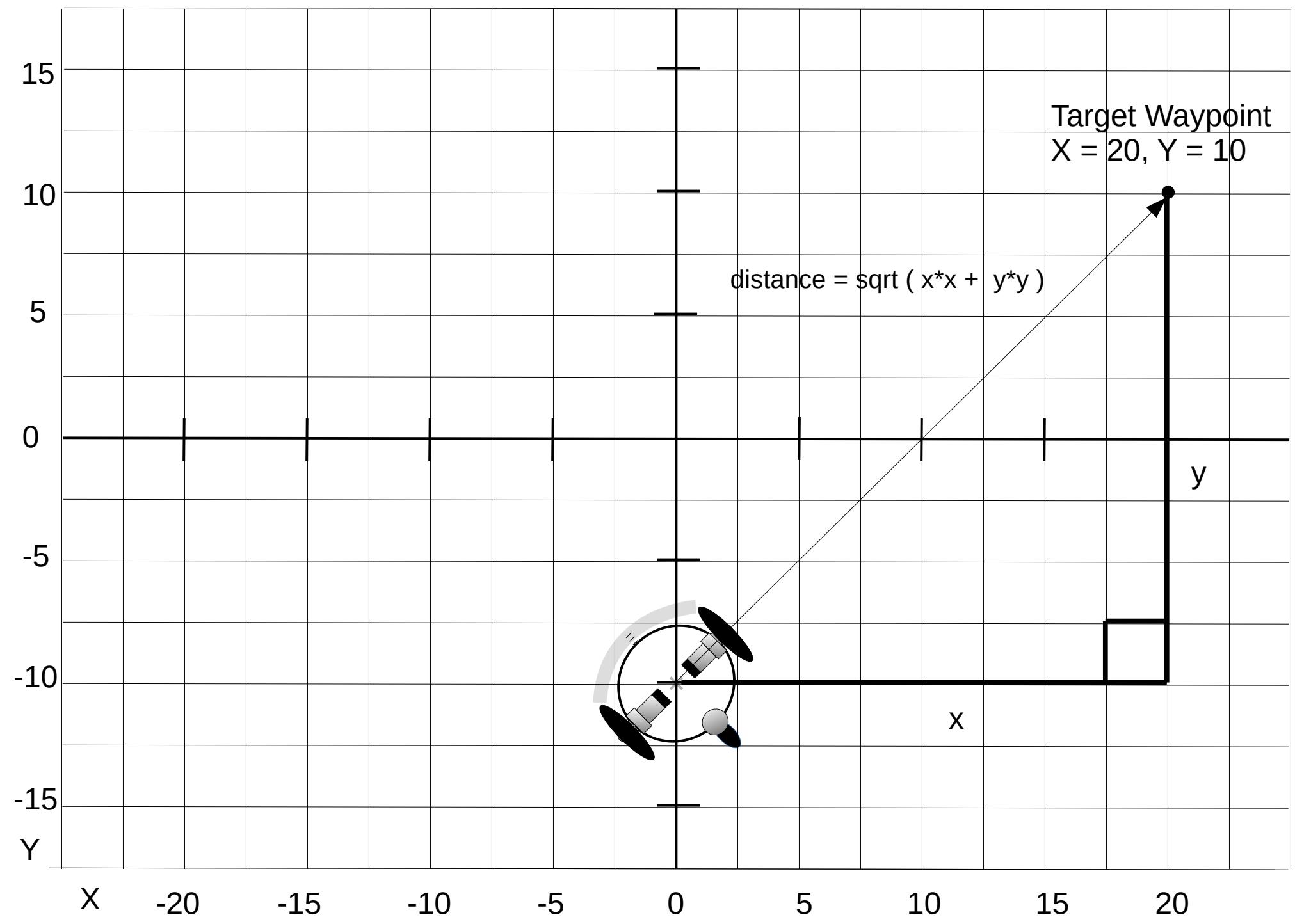
Where we want to go: Waypoints



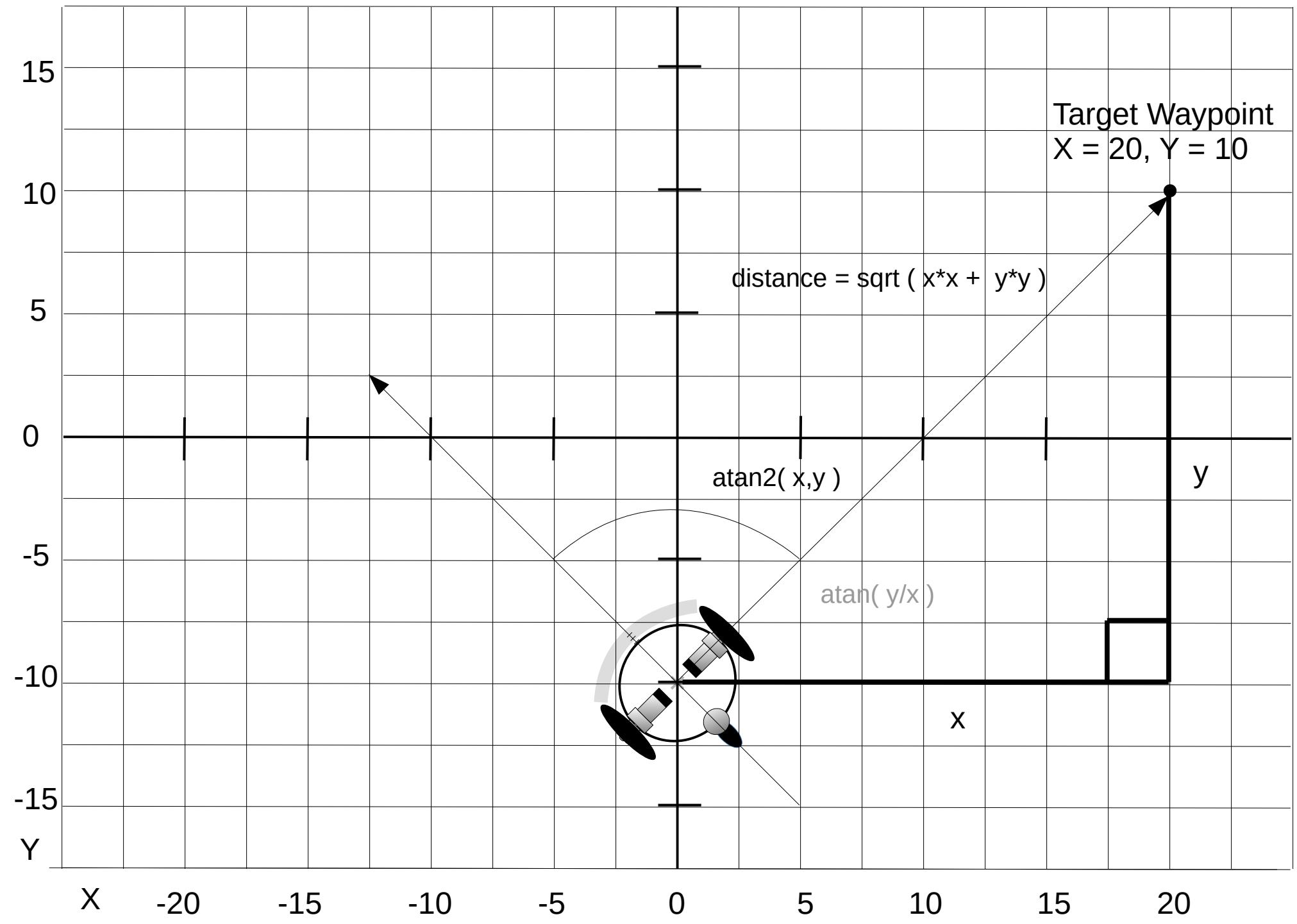
Pose: X = 0, Y = -10, Z = -45



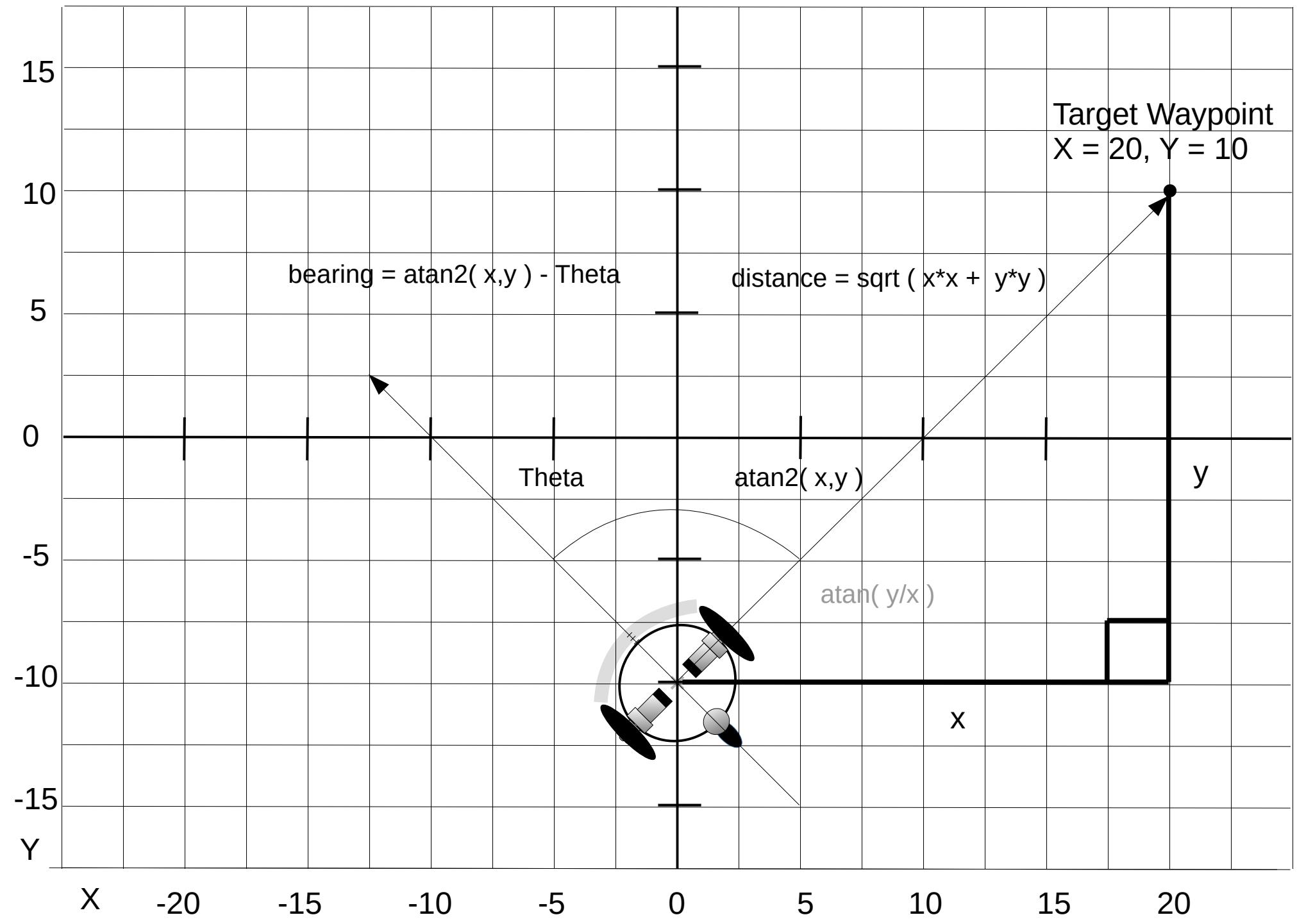
Pose: X = 0, Y = -10, Z = -45



Pose: X = 0, Y = -10, Z = -45



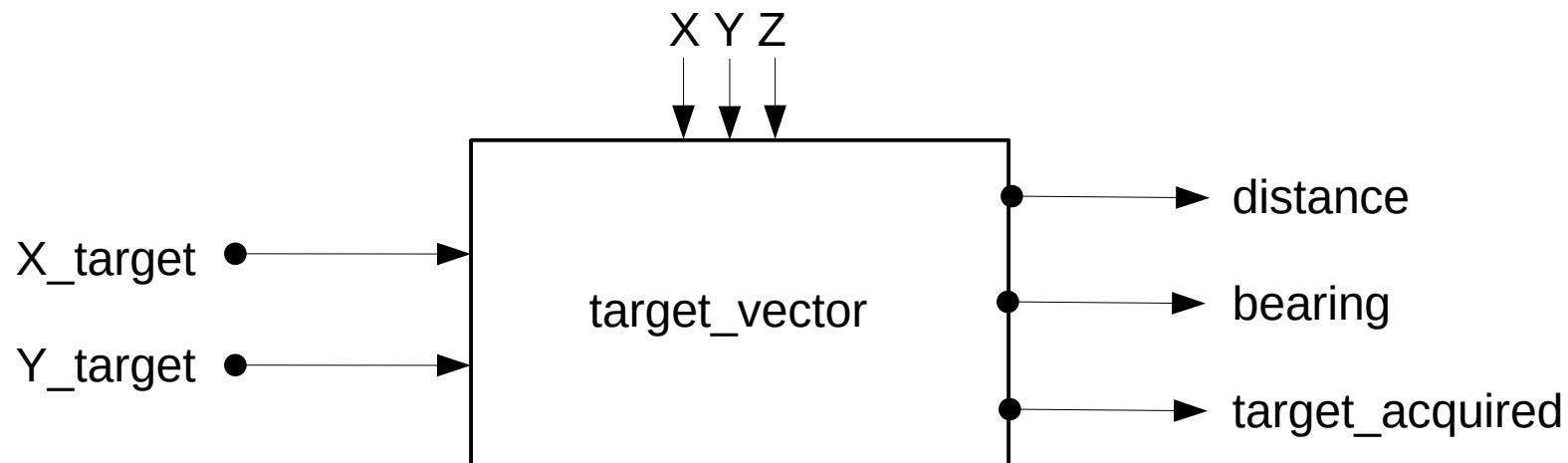
Pose: X = 0, Y = -10, Z = -45



B. A Navigation Virtual Sensor: target_vector()

POSE/Odometry provides at 25 Hz:

```
float X_position;      // x coordinate in inches  
float Y_position;      // y coordinate in inches  
float Theta;           // z coordinate in radians
```



Virtual Sensors

POSE/Odometry provides:

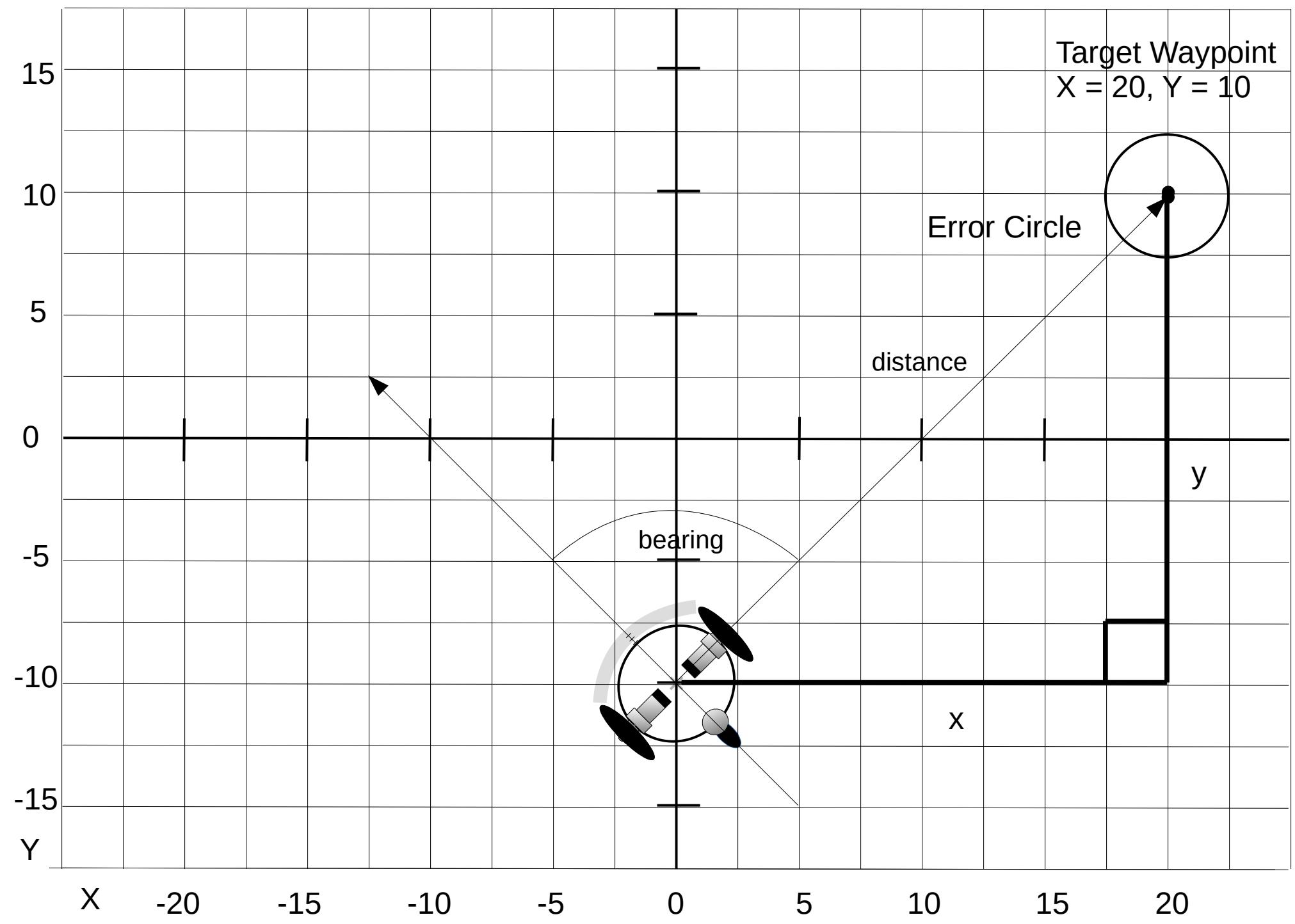
```
float X_position;      // x coordinate in inches  
float Y_position;      // y coordinate in inches  
float Theta;           // z coordinate in radians
```

```
int target_vector ( float X_target, Y_target, *distance, *bearing )  
{  
    float x, y;  
  
    x = X_target - X_position;  
    y = Y_target - Y_position;  
  
    *distance = sqrt ( ( x*x ) + ( y*y ) );  
    *bearing = atan2 ( x, y ) - Theta;  
  
    return target_acquired ( *distance );  
}
```

```
int target_acquired ( float distance )
{
    if ( distance == 0 ) return TRUE;
    else return FALSE;
}
```

```
int target_acquired ( float distance )
{
    if ( distance < ERR_CIRCLE ) return TRUE;
    else return FALSE;
}
```

Pose: X = 0, Y = -10, Z = -45



```
int target_acquired ( float distance, last_distance )
{
    if ((distance < ERR_CIRCLE) &&
        (distance > last_distance))
        return TRUE;
    else return FALSE;
}
```

Virtual Sensor

POSE/ODOMETRY provides:

```
float X_position;      // x coordinate in inches  
float Y_position;      // y coordinate in inches  
float Theta;           // z coordinate in radians
```

```
int target_vector ( float X_target, Y_target, *distance, *bearing )  
{  
    float x, y;  
    float last_distance = *distance;  
  
    x = X_target - X_position;  
    y = Y_target - Y_position;  
  
    *distance = sqrt ( ( x*x ) + ( y*y ) );  
    *bearing = atan2 ( x, y ) - Theta;  
  
    return target_acquired ( *distance, last_distance );  
}
```

C. Navigation behavior

Read Virtual Sensor:

call target_vector () and get current distance and bearing to target

Steering:

If bearing is positive, turn right.

If bearing is negative turn left.

if bearing is in DEADZONE, go straight.

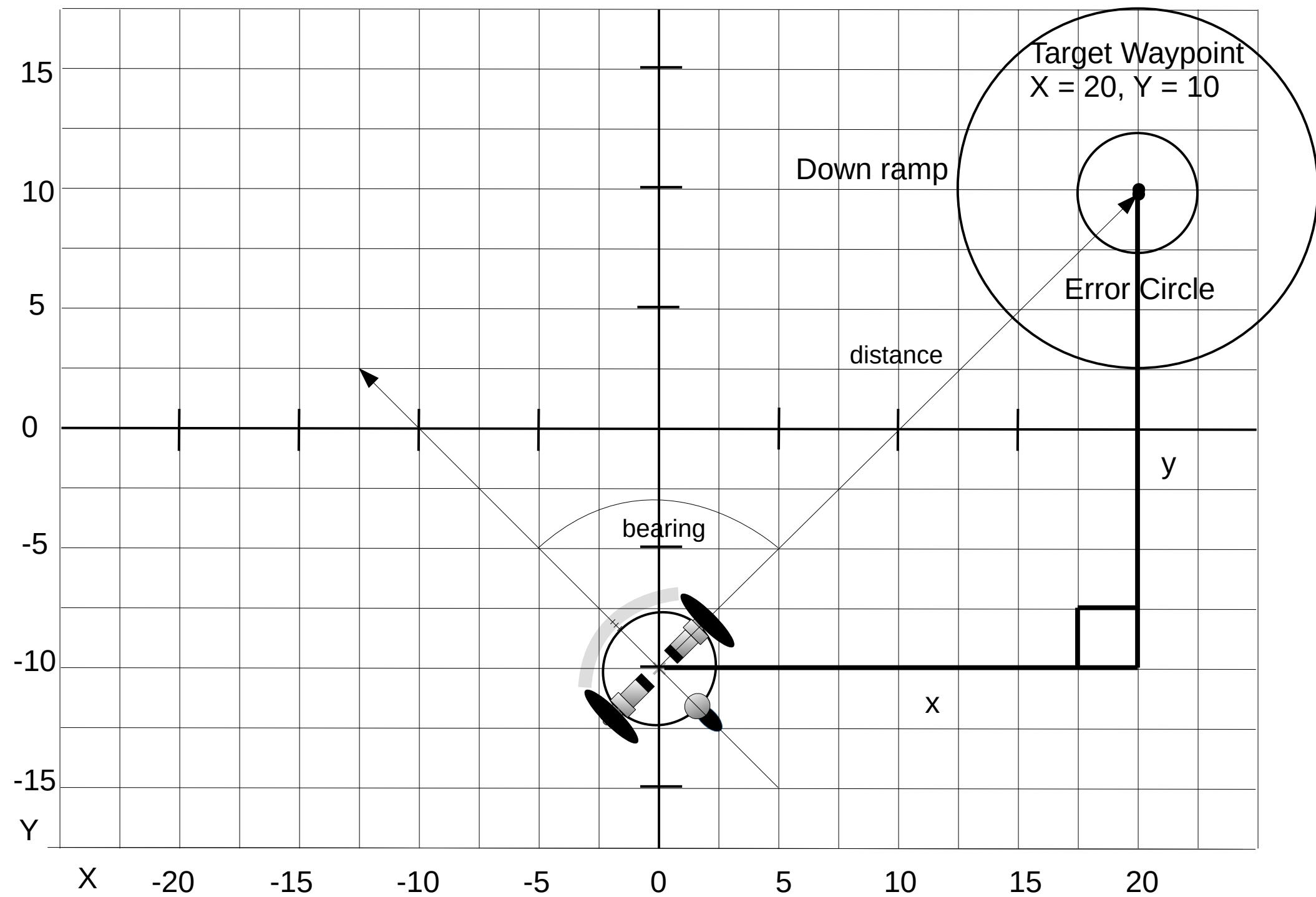
Speed:

if (distance > DOWNRAMP) velocity = p_user_velocity.

// else slow down approaching waypoint:

else velocity = p_user_velocity * (distance / DOWNRAMP)

Pose: X = 0, Y = -10, Z = -45




```
void navigate_behavior ( void )
{
    extern NAVLIST *navlist, *navlist_begin, *navlist_end;      // X,Y list

    float dist, bear;

    if ( navlist ) {

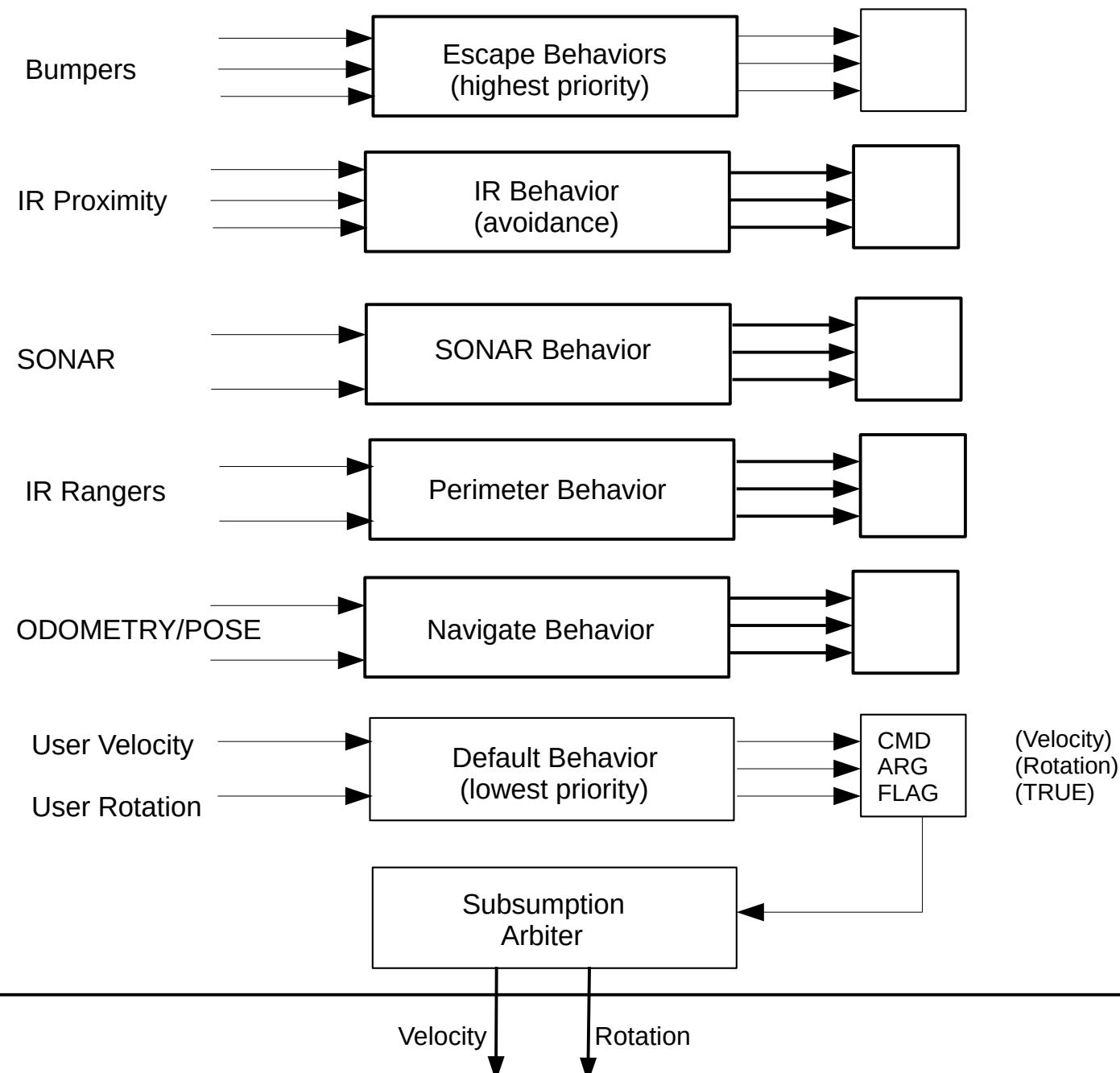
        navigate.flag = TRUE;

        if ( ( navigate_target ( navlist->X, navlist->Y ) ) == TRUE ) {

            navlist++;
            if ( navlist > navlist_end ) {
                if ( p_nav_repeat ) navlist = navlist_begin;
                else navlist = 0;
            }

            if ( ( navlist ) && ( p_stop_enable ) ) {      // stopped?
                target_vector ( navlist->X, navlist->Y, &dist, &bear );
                rotate ( bear );
            }
        }
    } else {
        navigate.flag = FALSE;
    }
}
```

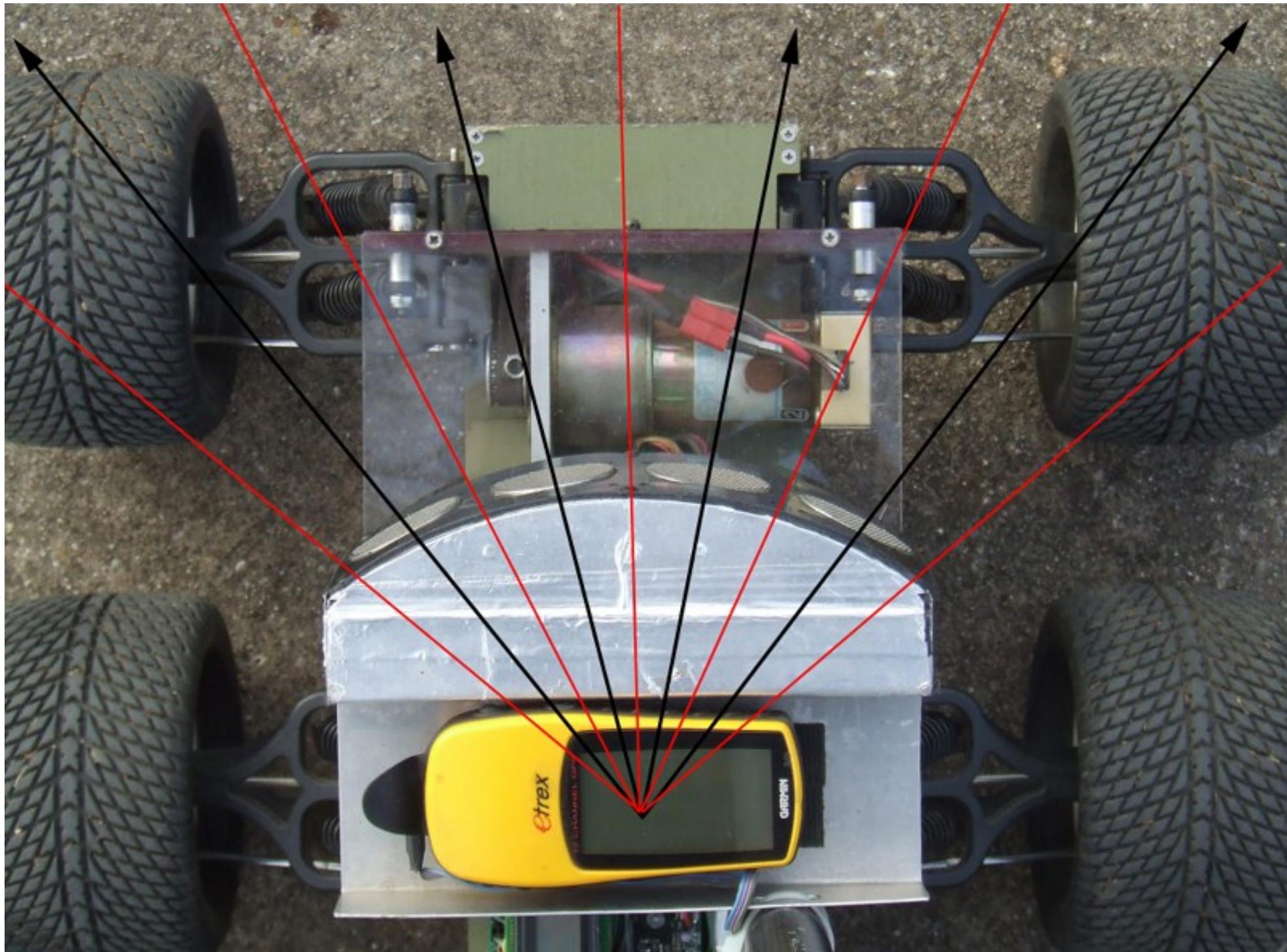
25 Hz Control Loop



Demo 5

Live

RCAT: navigation and obstacle avoidance



Demo 6

Video
jBot: Hat trick

Jbot's Control Loop

```
void sensors (ASIZE ignored)
{
    TSIZE t;

    t = sysclock;
    while (1) {

        /* utilities */
        do_speedometer();
        odometers();

        /* behaviors */
        prowl_task();
        bump_task();
        deadman_task();
        navigate_task();
        escape_task();
        obstacle_task();
        perimeter_task();
        navmode_task();
        seek_task();
        xlate_task();

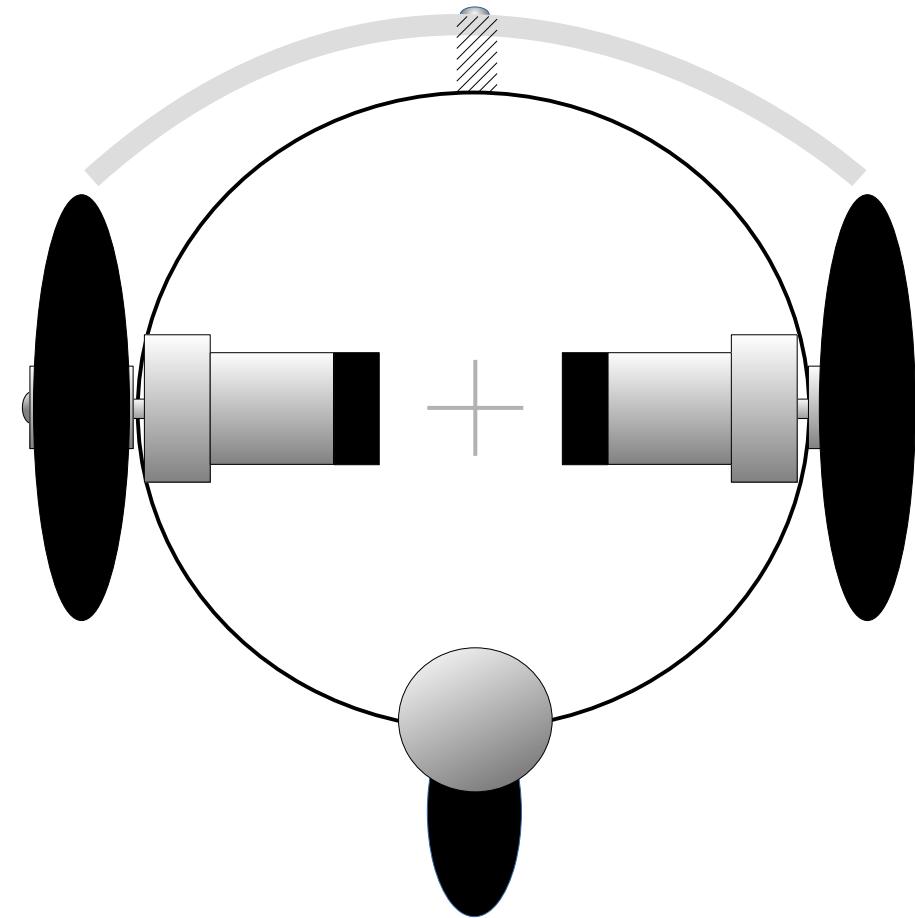
        /* arbitrate and call motorcmd(); */
        arbitrator();

        /* at user rate or 25 Hz */
        if (p_user_sensor_rate) wake_after(p_user_sensor_rate);
        else PERIOD(&t,40);
    }
}
```

Demo 7

Video
jBot: Cones

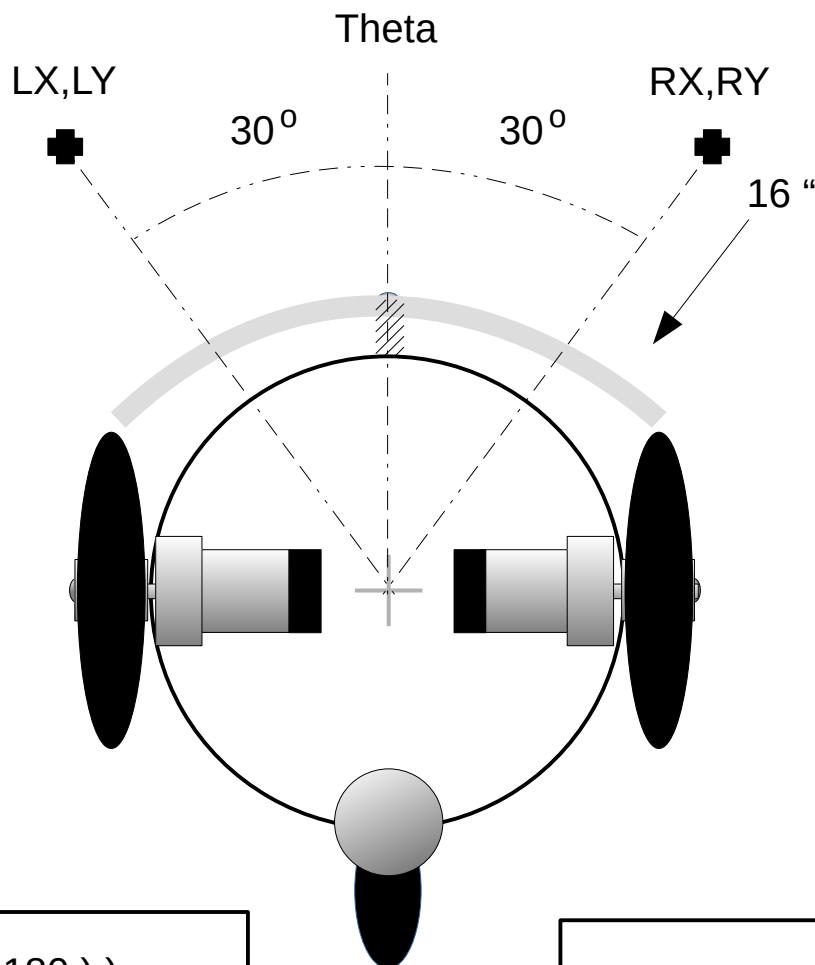
III. Advanced Navigation Modes



III. Advanced Navigation Modes

- A. Boundary Detector
- B. Slip-Stuck Detector
- C. Concave Singularity Detector
(“Minnow Trap”)

A. Virtual Boundary Detectors



$$L_angle = \Theta - (30 * (\pi / 180))$$

$$LX = X_position + (\sin(L_angle) * 16)$$
$$LY = Y_position + (\cos(L_angle) * 16)$$

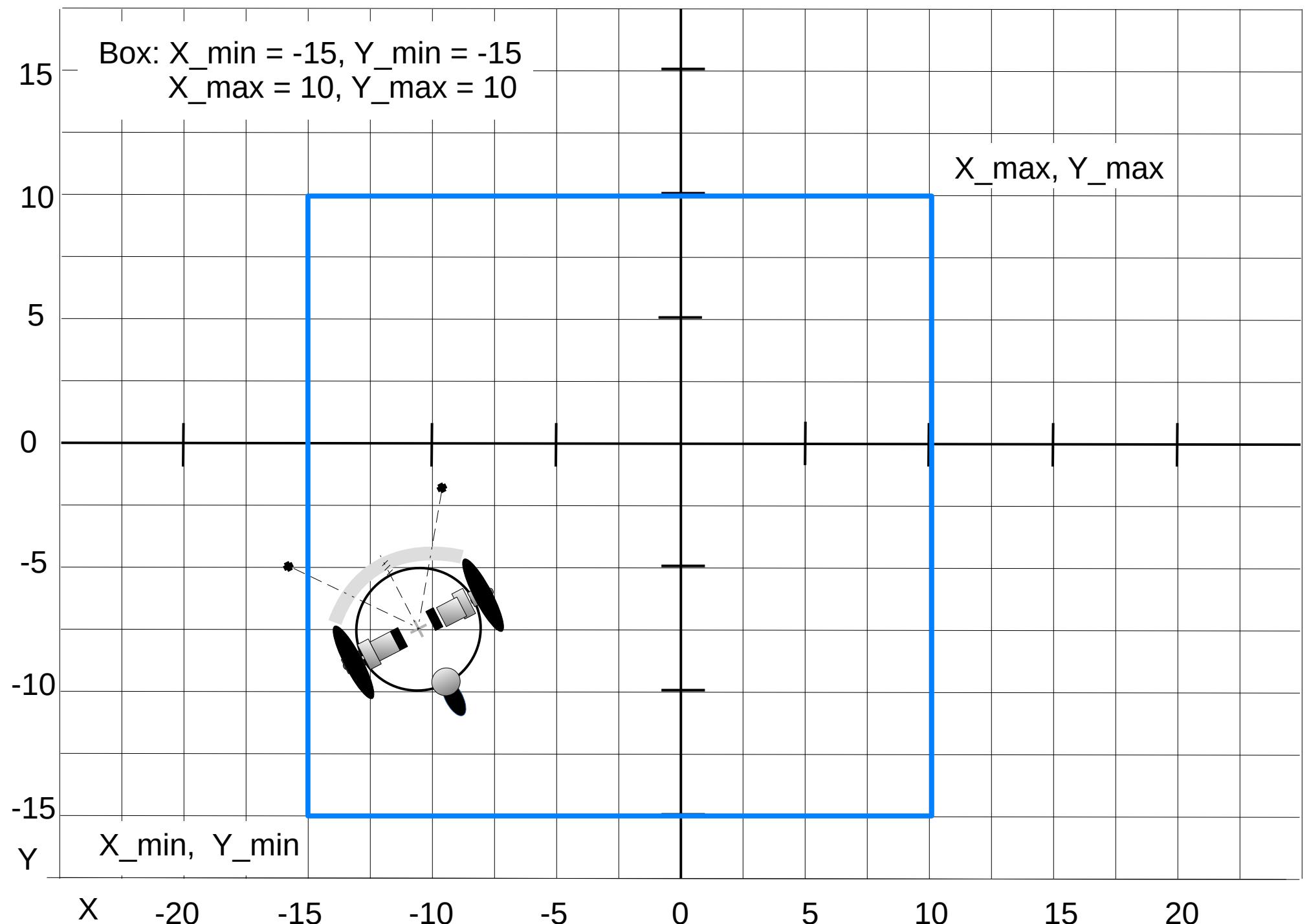
$$R_angle = \Theta + (30 * (\pi / 180))$$

$$RX = X_position + (\sin(R_angle) * 16)$$
$$RY = Y_position + (\cos(R_angle) * 16)$$

Box: X_min = -15, Y_min = -15
X_max = 10, Y_max = 10

X_max, Y_max

X_min, Y_min



Virtual Sensors: Boundary Detectors

```
int boundary ( float x, y )
{
    extern BOX box;

    if ( ( x < box.xmin ) || ( x > box.xmax ) )
        return TRUE;

    if ( ( y < box.ymin ) || ( y > box.ymax ) )
        return TRUE;

    return FALSE;
}
```

Demo 8

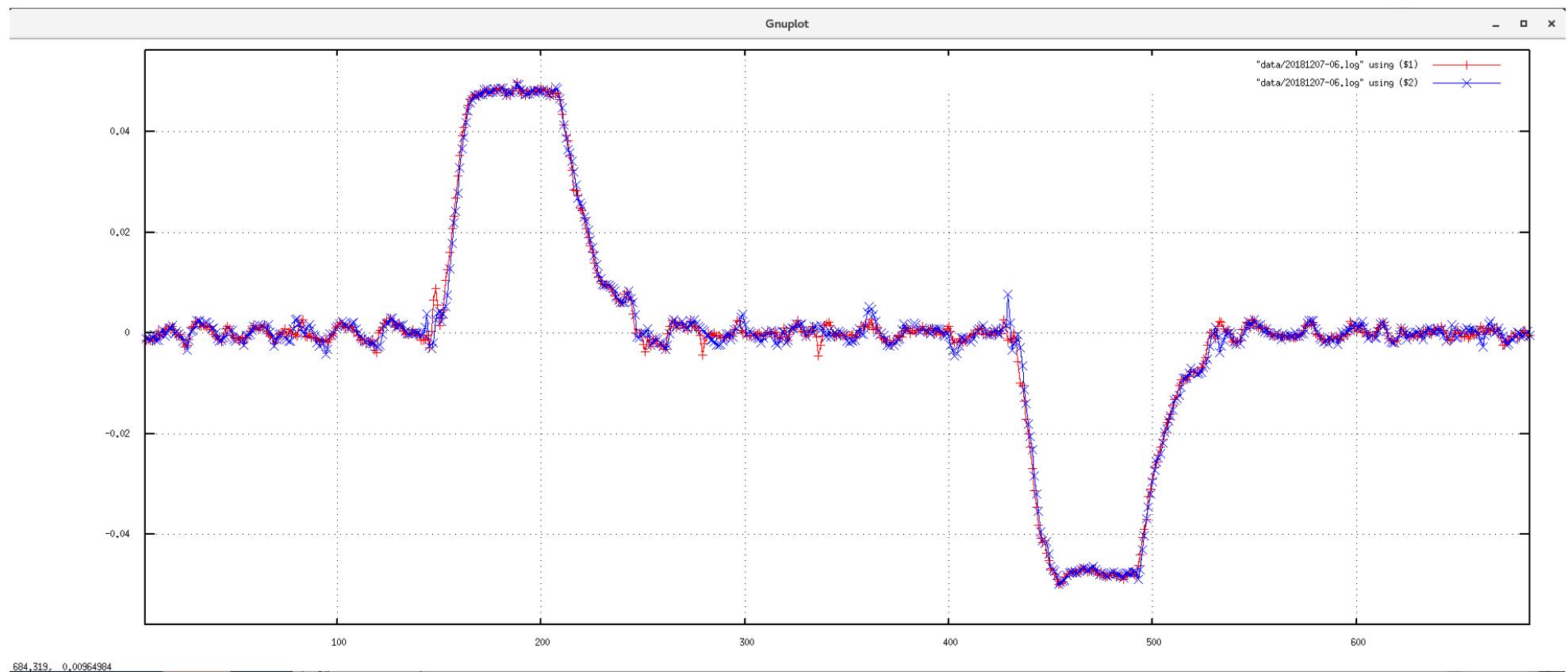
Live
RCAT: Boundary Detection

Video
RCAT: Hallway Boundary

B. Virtual Slip/Stuck Detector

Detecting and Getting Unstuck

wheel rate vs. gyro rate



```
int dif_count = 0;

void slip-stuck_behavior ( void )
{
    float rate_dif = wheel_rate - gyro_rate;

    if ( ( rate_dif > MAXRATEDIF ) || ( rate_dif < -MAXRATEDIF ) ){

        dif_count++;
        if ( dif_count > MAXDIFCOUNT ) {
            trigger_escape();
        }

    } else dif_count = 0;
}
```

Demo 9

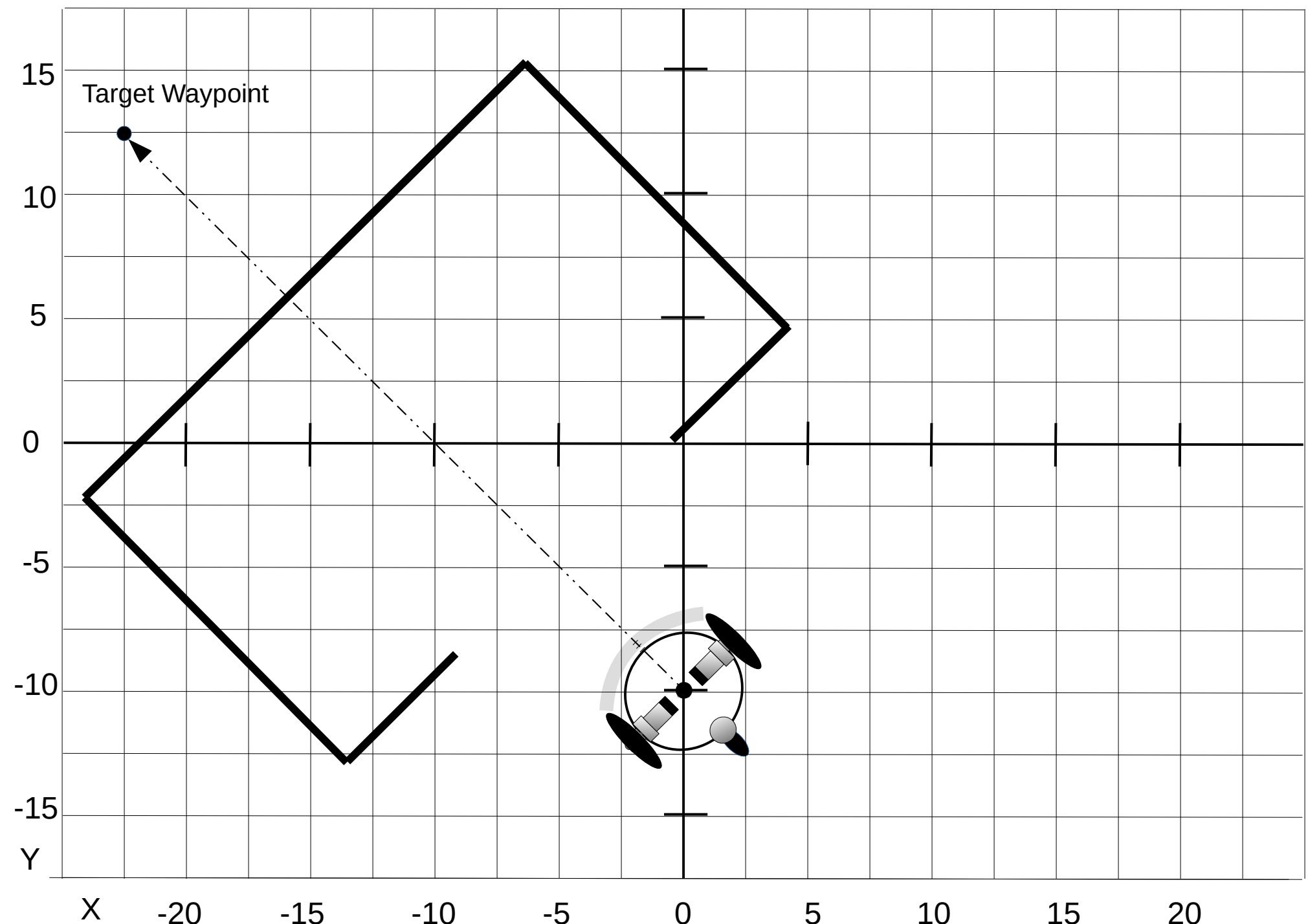
Live
RCAT: Virtual bumper

Video
jBot: Getting Unstuck

C. Virtual Concave Singularity Detector

Concave Waypoint Singularity:
(Minnow Trap)

Pose: X = 0, Y = -10, Z = -45



D10

Demo 10

Video
LegoBot: waypoint trap

```
void navmode_behavior ( void )
{
    extern int perimeter_enable;      // enable perimeter following

    if ( perimeter_enable == FALSE {
        if ( bearing == 180 ) perimeter_enable = TRUE;
    }

    if ( perimeter_enable == TRUE ) {
        if ( bearing == 0 ) perimeter_enable = FALSE;
    }
}
```



Demo 11

Video
jBot @ TI