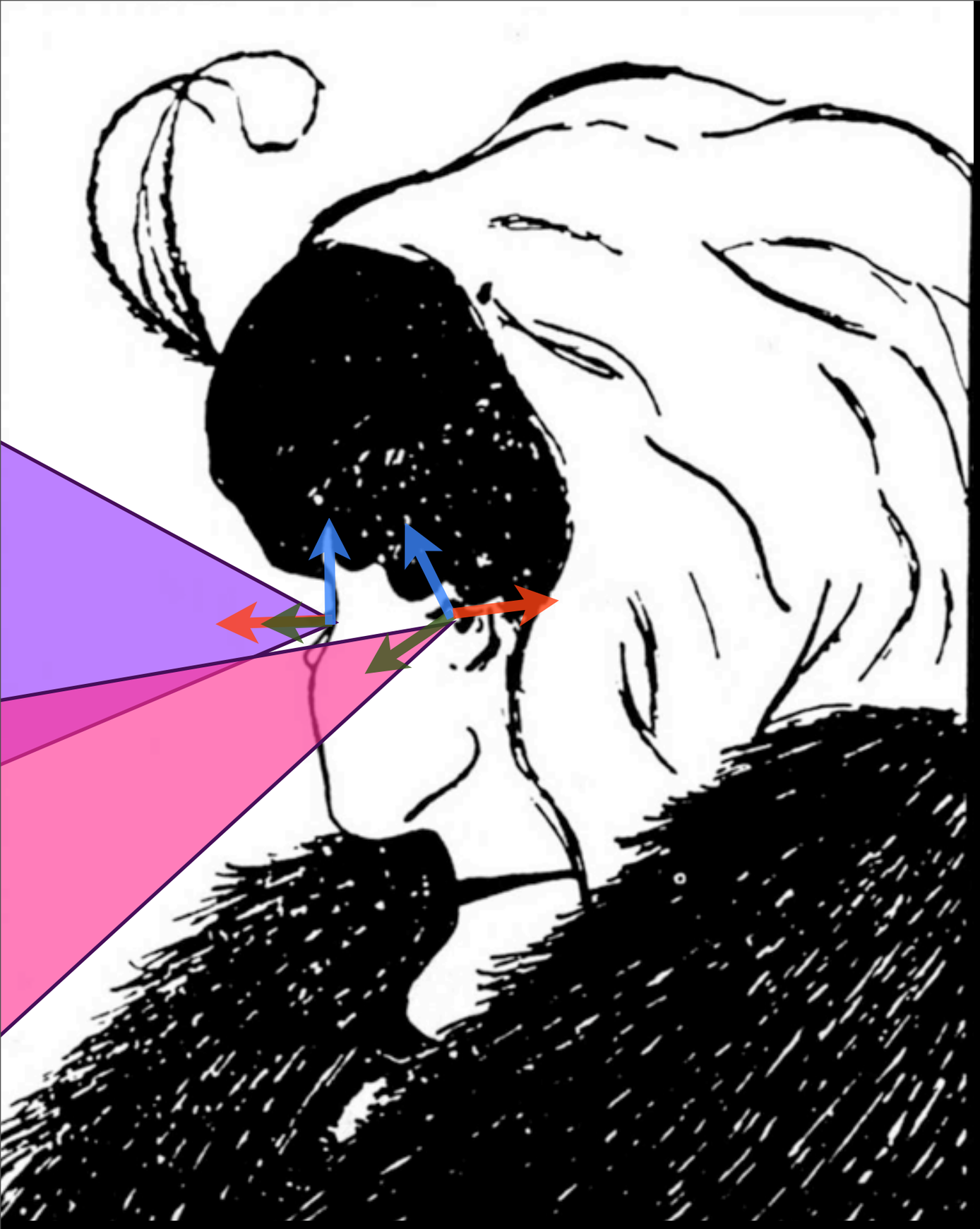
Topic 10

Coordinate Spaces and Transforms

It depends on your perspective



consider "left eye coordinates"

Topic 10

Coordinate Spaces and Transforms

It depends on your perspective

JointTrajectory example for planning project

File Edit View History Bookmarks Tools Help

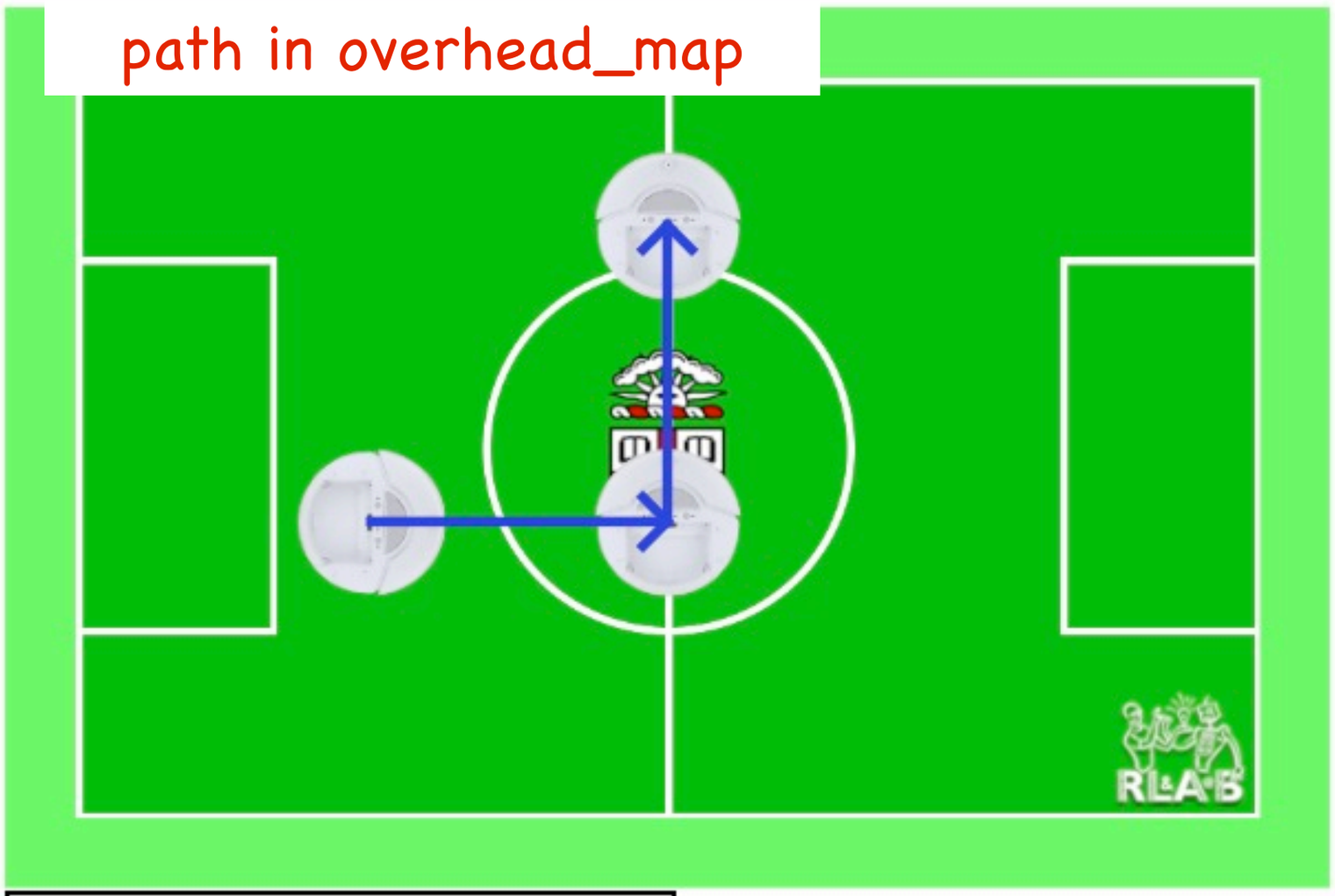
file:///opt/ros/cturtle/ros/brown-ros-pkg/overhead_map/overhead_m

2d transforms

Feedback

7. Compound ... | Global variabl... | Google Docs -... | Problem loadi... | brown-ros-pk.

path in overhead_map



JointTrajectory definition

JointTrajectory message:

Header header
  uint32 seq
  time stamp
  string frame_id
string[] joint_names
JointTrajectoryPoint[] points
  float64[] positions
  float64[] velocities
  float64[] accelerations
  duration time_from_start

Done

JointTrajectory example for planning project

File Edit View History Bookmarks Tools Help

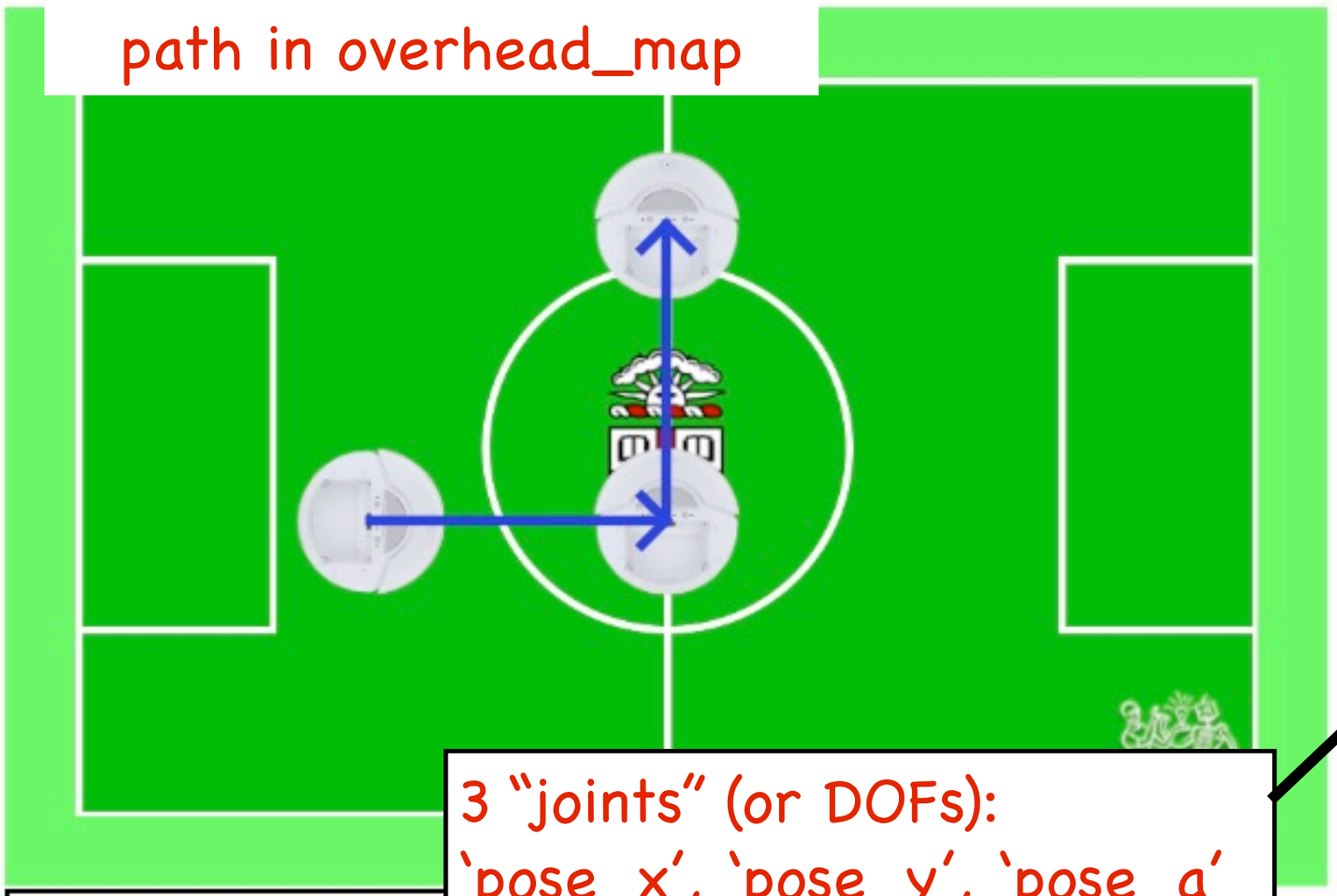
file:///opt/ros/cturtle/ros/brown-ros-pkg/overhead_map/overhead_m

2d transforms

Feedback

7. Compound ... | Global variabl... | Google Docs -... | Problem loadi... | brown-ros-pk.

path in overhead_map



3 "joints" (or DOFs):
'pose_x', 'pose_y', 'pose_a'

each "point" specifies intermediate DOF values at a "time_from start"

JointTrajectory definition

JointTrajectory message:

```
Header header
    uint32 seq
    time stamp
    string frame_id
string[] joint_names
JointTrajectoryPoint[] points
    float64[] positions
    float64[] velocities
    float64[] accelerations
    duration time_from_start
```

Done

use 0 for velocities and accelerations, sequence number for duration

3 "joints" (or DOFs): 'pose_x', 'pose_y', 'pose_a'

publish as topic 'DOF3PlanarPath'

```
...
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
...
def trajmsg_test():
    pub = rospy.Publisher('DOF3PlanarPath', JointTrajectory)
    traj = JointTrajectory();
    traj.joint_names = ('pose_x', 'pose_y', 'pose_a');
    trajpoint = JointTrajectoryPoint();
    trajpoint.positions = (1,1,0);
    trajpoint.velocities = (0,0,0);
    trajpoint.accelerations = (0,0,0);
    trajpoint.time_from_start = roslib.rostime.Duration(1);
    traj.points.append(trajpoint);
    trajpoint = JointTrajectoryPoint();
    trajpoint.positions = (2,1,pi/2);
    trajpoint.velocities = (0,0,0);
    trajpoint.accelerations = (0,0,0);
    trajpoint.time_from_start = roslib.rostime.Duration(2);
    traj.points.append(trajpoint);
    trajpoint = JointTrajectoryPoint();
    trajpoint.positions = (2,2,pi/2);
    trajpoint.velocities = (0,0,0);
    trajpoint.accelerations = (0,0,0);
    trajpoint.time_from_start = roslib.rostime.Duration(3);
    traj.points.append(trajpoint)
...
    while not rospy.is_shutdown():
        pub.publish(traj)
```

each "point" specifies intermediate DOF values at a "time_from start"

Displaying path in overhead web visualization

import om_msgs messages

specify objects in Overhead_Map_Objs in DOF3PlanarPath handler

do not forget to run rosjs

subscribe to DOF3PlanarPath, publish overhead_map_objs

```
...
from om_msgs.msg import Overhead_Map_Objs, Overhead_Map_Obj
...
def processPath(DOF3PlanarPath):
        global curpathdrawobjs;
        drawobjs = Overhead_Map_Objs();
        for i in range(len(DOF3PlanarPath.points)-1):
                drawobj = Overhead_Map_Obj();
                drawobj.name = 'arrow';
                drawobj.tuple =
                  [DOF3PlanarPath.points[i].positions[0],
                  DOF3PlanarPath.points[i].positions[1],
                  DOF3PlanarPath.points[i+1].positions[0],
                  DOF3PlanarPath.points[i+1].positions[1]];
                drawobjs.objs.append(drawobj);
        curpathdrawobjs = drawobjs;
...
def trajmsg_display():
    pub = rospy.Publisher('overhead_map_objs', Overhead_Map_Objs);
    rospy.Subscriber('DOF3PlanarPath', JointTrajectory, processPath);
    ...
    while not rospy.is_shutdown():
        pub.publish(curpathdrawobjs)
...
```

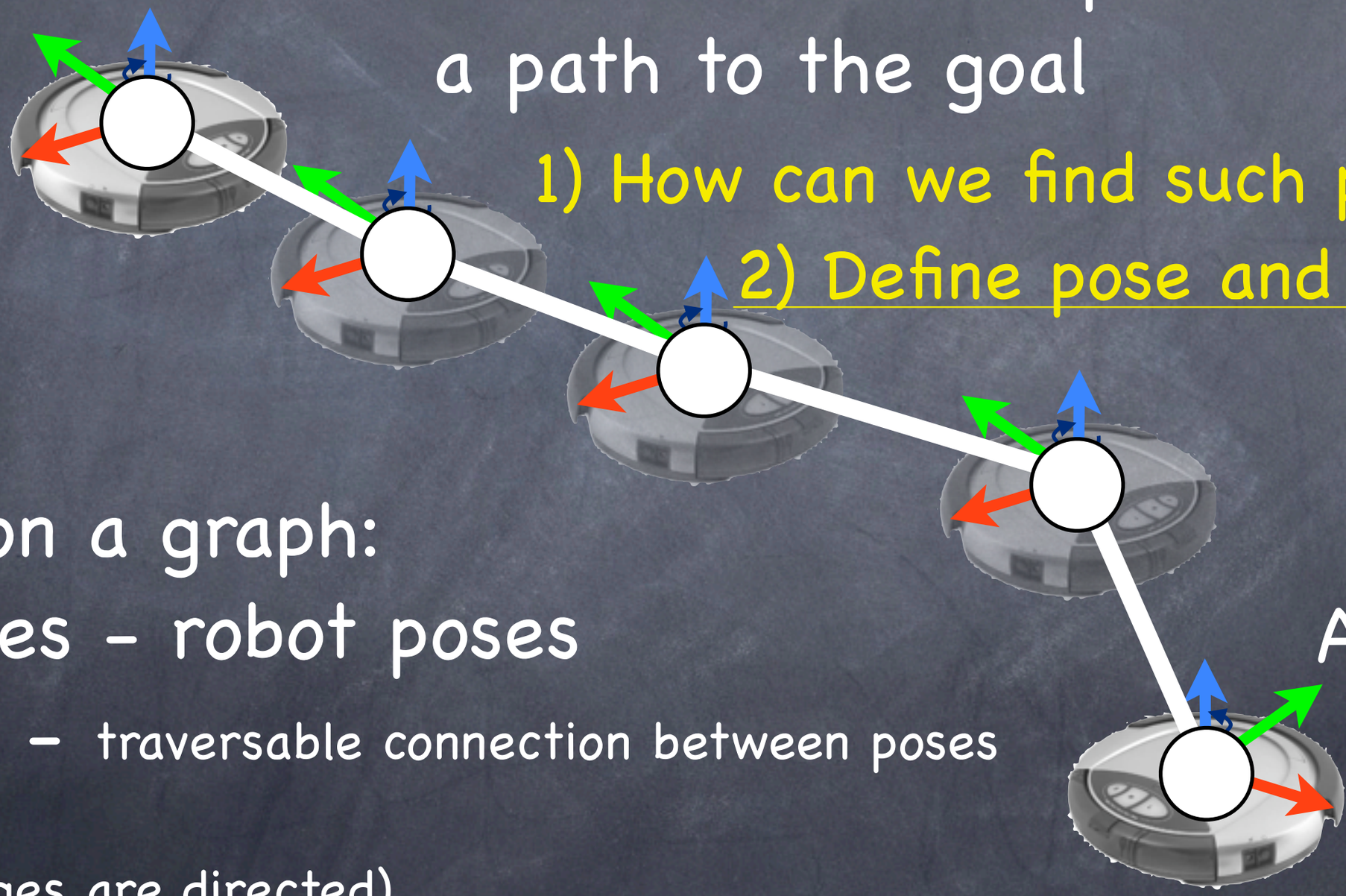
Path Planning

B: Goal

Find intermediate poses forming a path to the goal

1) How can we find such paths?

2) Define pose and controls?



A: Start

Path on a graph:

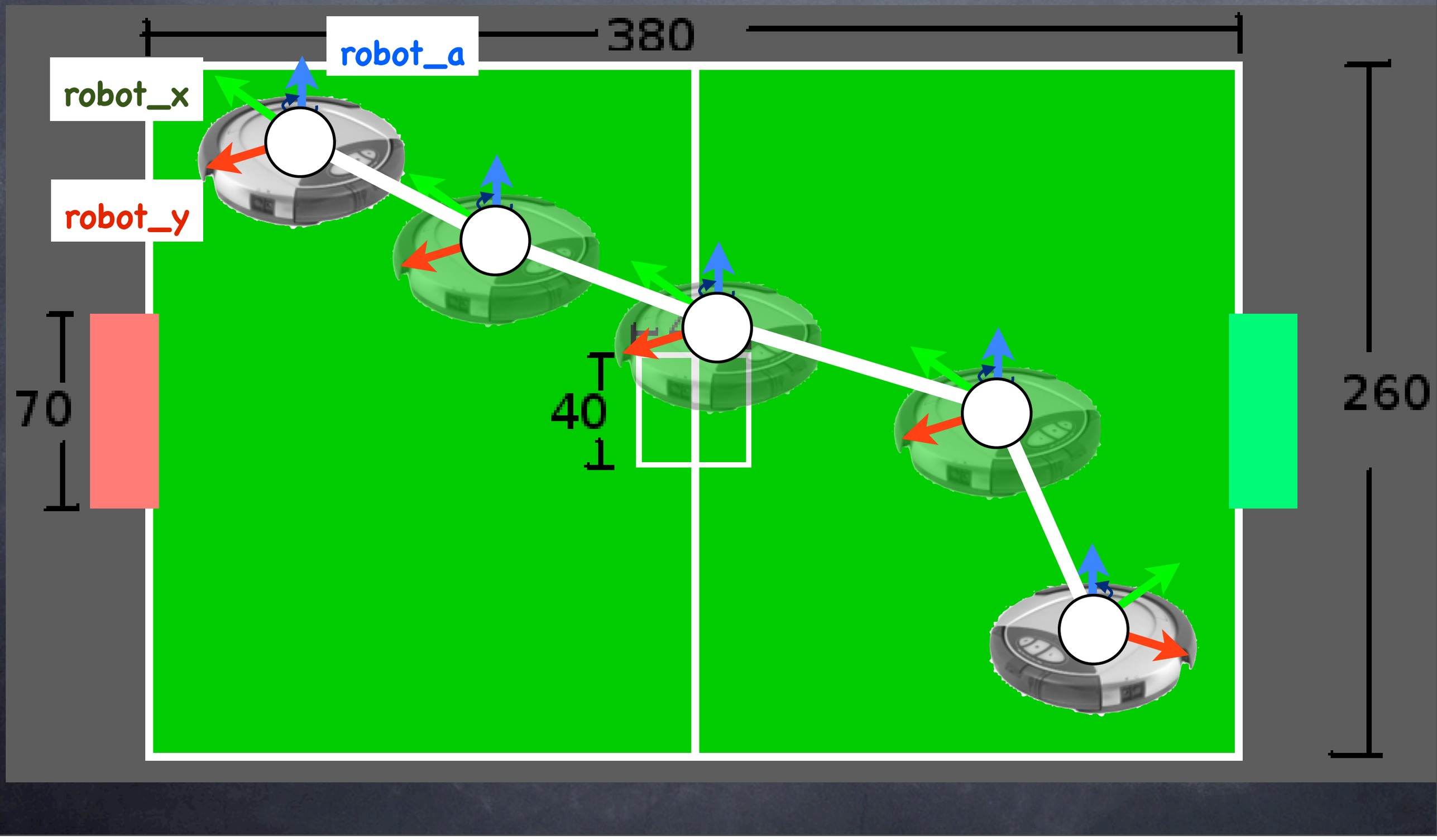
vertices - robot poses

edges - traversable connection between poses

(note edges are directed)

1) path planning: sequence of intermediate poses

2) Define pose and controls?

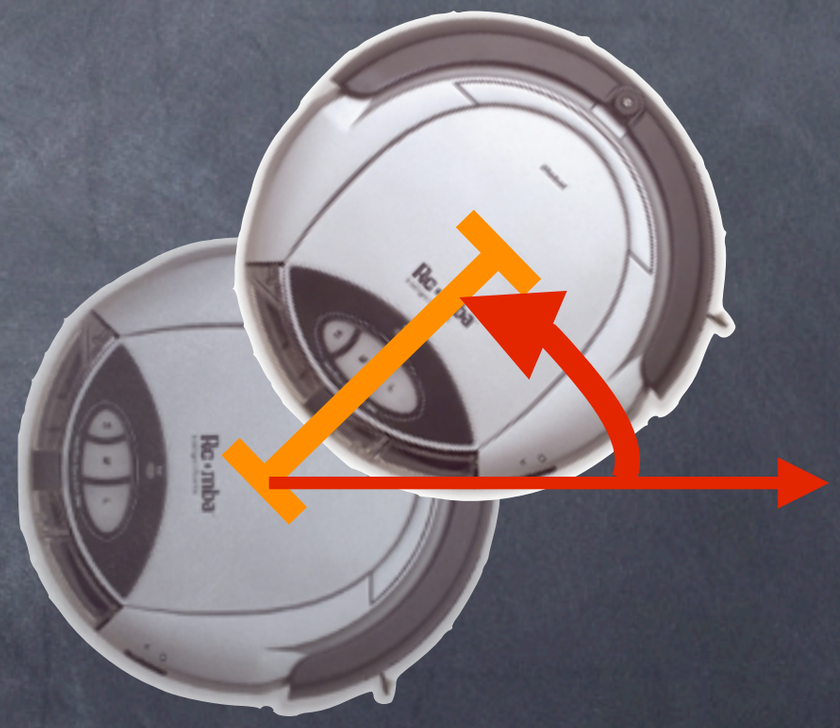


Tangent: Integrating Odometry?

- Path planning assumes we know location of robot
- Suppose no overhead localization?
- Could we use the robot's odometry?
- Can we plot robot's movement on field using odometry?

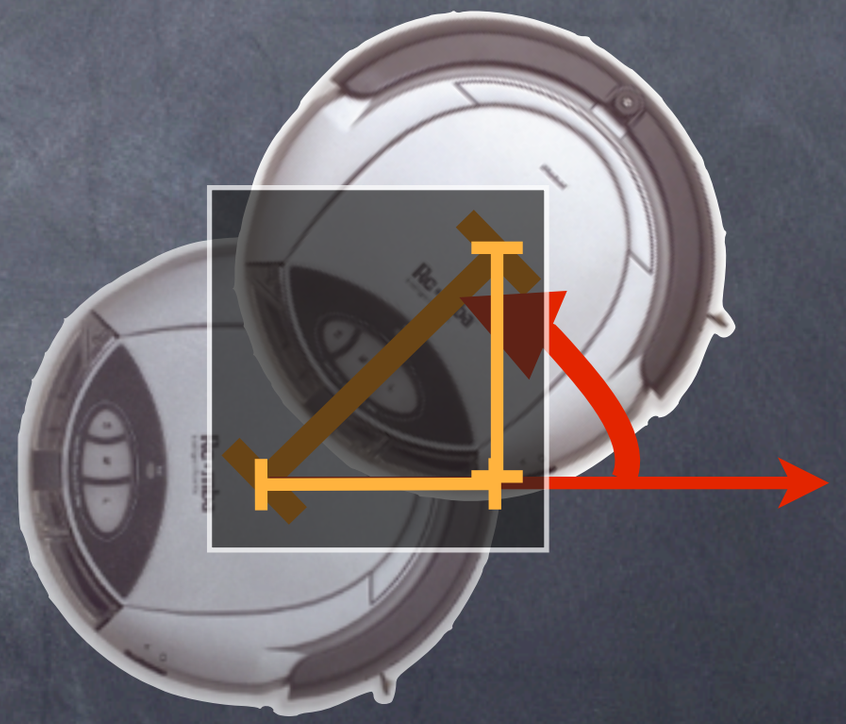
Tangent: Integrating Odometry?

- irobot_create_2_1 publishes
  - **angle**: amount of turn since last update
  - **distance**: distance traveled since last update
- can we use these to "dead reckon" estimate of robot pose (x,y,a)?



Tangent: Integrating Odometry?

- A simple answer:
  - pose_x → pose_x + **distance** * cos(**angle**)
  - pose_y → pose_y + **distance** * sin(**angle**)
  - pose_a → (pose_a + **angle**) % 2pi



What happens next?

At the next update,
the robot drives forward

Where will odometry update
put the robot's pose?

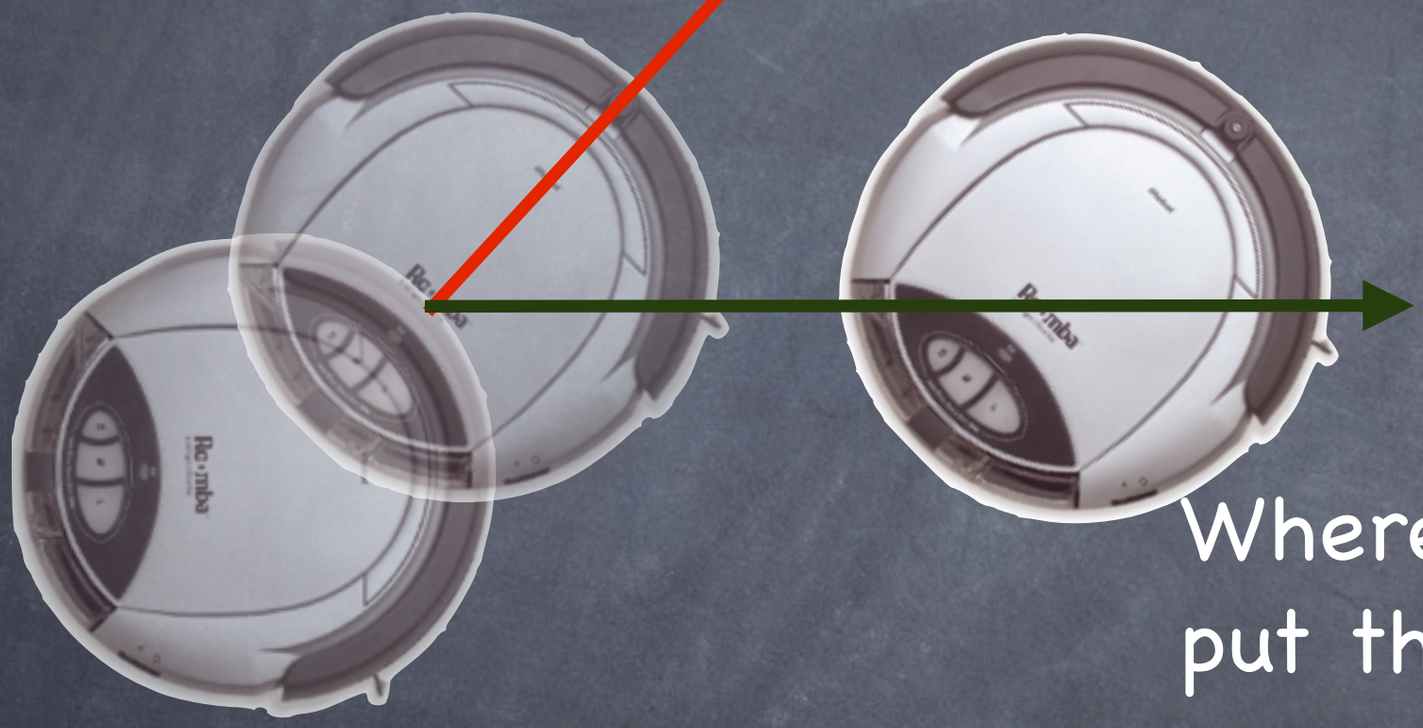
pose_x → pose_x + **distance** * cos(**angle**)

pose_y → pose_y + **distance** * sin(**angle**)

pose_a → (pose_a + **angle**) % 2pi

What happens next?

forward in world coordinates



At the next update, the robot drives forward

forward in robot coordinates

Where will odometry update put the robot's pose?

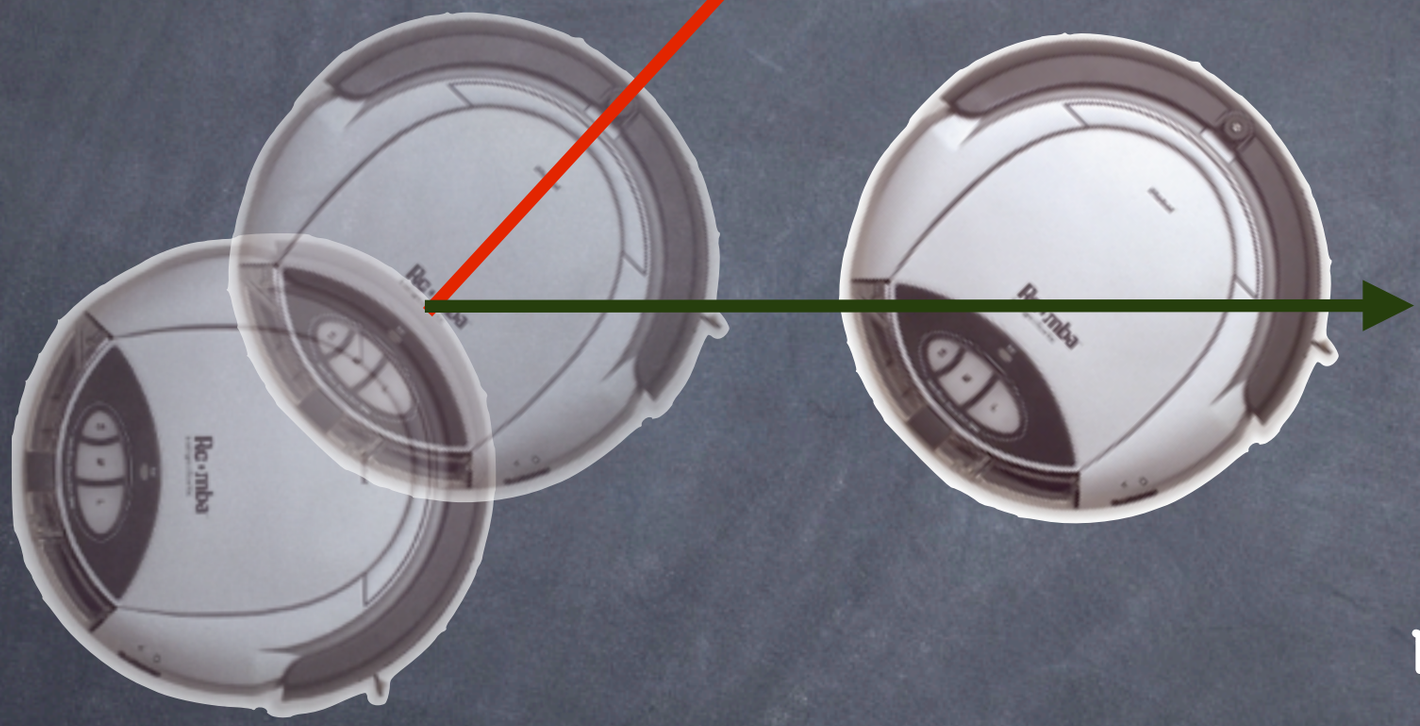
pose_x → pose_x + **distance** * cos(**angle**)

pose_y → pose_y + **distance** * sin(**angle**)

pose_a → (pose_a + **angle**) % 2pi

What happens next?

forward in world coordinates



At the next update, the robot drives forward

forward in robot coordinates

Need to account for rotation w.r.t. the world

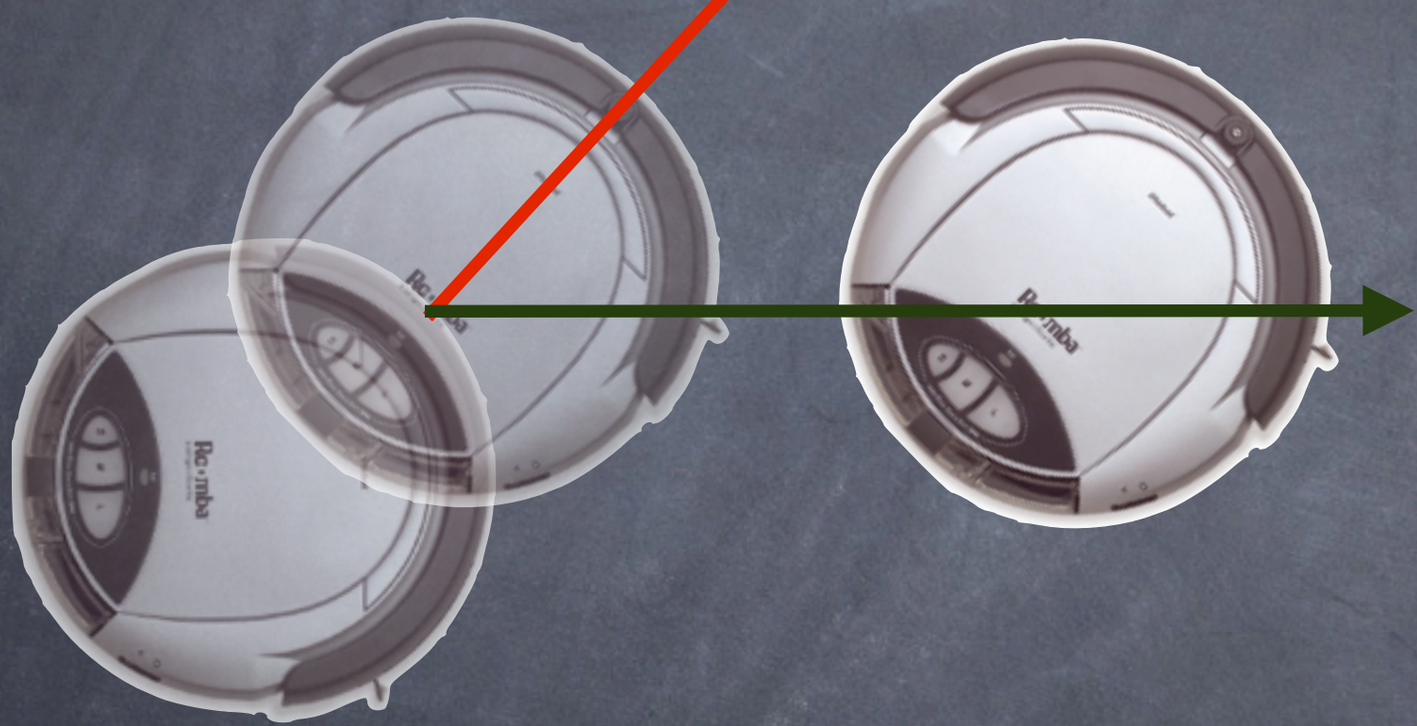
pose_x → pose_x + **distance** * cos(**angle**)

pose_y → pose_y + **distance** * sin(**angle**)

pose_a → (pose_a + **angle**) % 2pi

How to account for robot orientation in the world?

Problem: the robot's translation is dependent on its rotation



pose_x → pose_x + **distance** * cos(**angle**)

pose_y → pose_y + **distance** * sin(**angle**)

pose_a → (pose_a + **angle**) % 2pi

How to account for robot orientation in the world?

Express odometry as vector robot coordinates

$$odom_{robot} = \begin{bmatrix} \textbf{distance} * \cos(\textbf{angle}) \\ \textbf{distance} * \sin(\textbf{angle}) \end{bmatrix}$$

How to account for robot orientation in the world?

Rotate by pose angle in world coordinates ...

$odom_{world}$ =

... * rotate(pose_a) * $odom_{robot}$

How to account for robot orientation in the world?

and translate by pose offset in world coordinates:

$odom_{world}$ =

translate(pose_x,pose_y) * rotate(pose_a) * $odom_{robot}$

How to account for robot orientation in the world?

Now, the direct odometry update can be applied

$pose_{world\_x} \rightarrow pose_{world\_x} + odom_{world\_x}$

$pose_{world\_y} \rightarrow pose_{world\_y} + odom_{world\_y}$

$pose\_a \rightarrow (pose\_a + \mathbf{angle}) \% 2pi$

but wait, how do we actually compute $odom_{world}$?

Now, the direct odometry update can be applied

$pose_{world}\_x \rightarrow pose_{world}\_x + odom_{world}\_x$

$pose_{world}\_y \rightarrow pose_{world}\_y + odom_{world}\_y$

$pose\_a \rightarrow (pose\_a + \mathbf{angle}) \% 2pi$

but wait, how do we actually compute $odom_{world}$?

Now, the direct odometry update can be applied

$pose_{world}\_x \rightarrow pose_{world}\_x + odom_{world}\_x$

$pose_{world}\_y \rightarrow pose_{world}\_y + odom_{world}\_y$

$pose\_a \rightarrow (pose\_a + \mathbf{angle}) \% 2pi$

where,
tx = pose_x
ty = pose_y
theta = pose_a

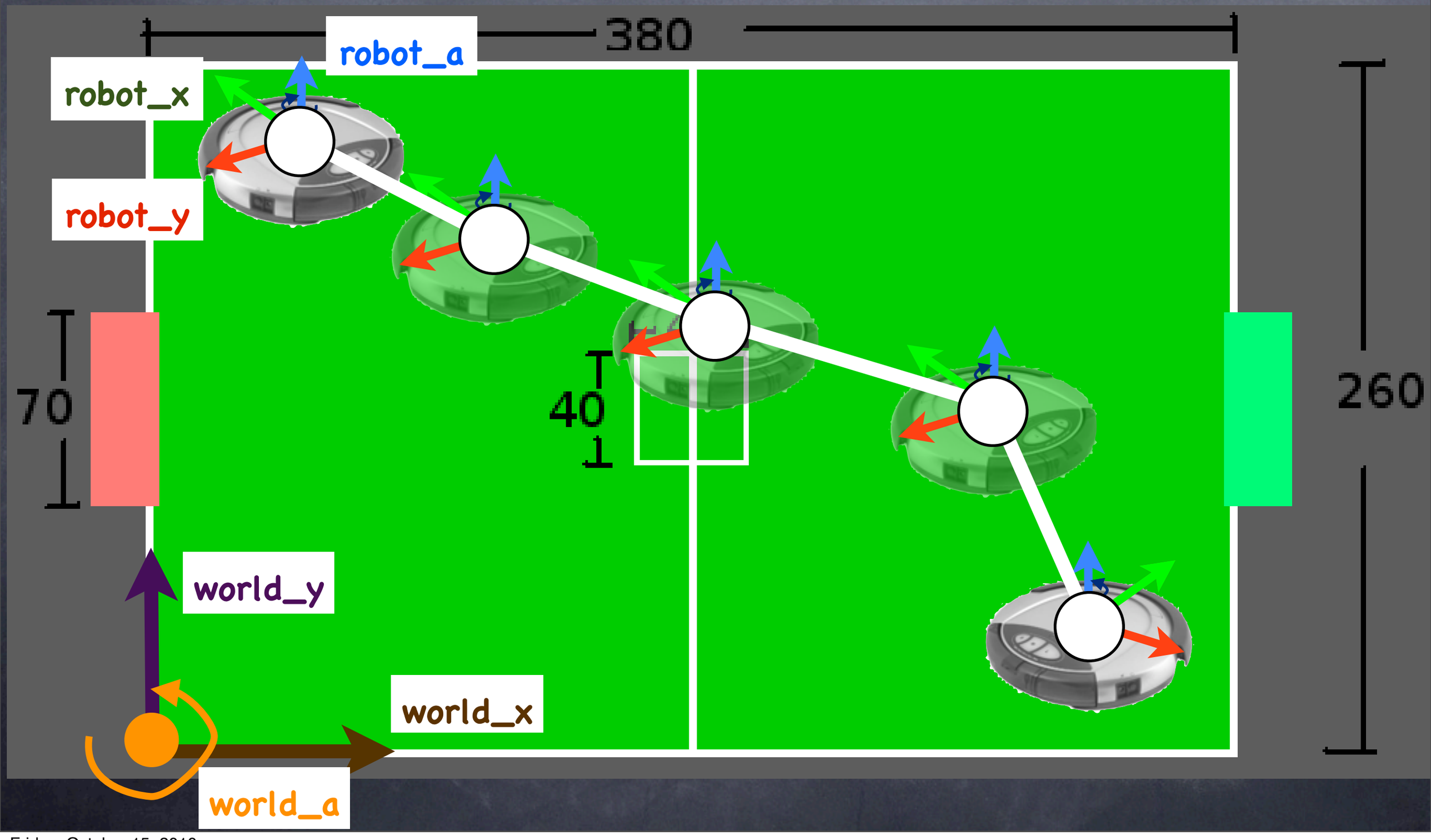
$$\begin{bmatrix} odom_{world}\_x \\ odom_{world}\_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} odom_{robot}\_x \\ odom_{robot}\_y \\ 1 \end{bmatrix}$$

$odom_{world}$ =
translate(pose_x,pose_y) * rotate(pose_a) * $odom_{robot}$

1) path planning: sequence of intermediate poses

2) Define pose (3DOF position/orientation on field)

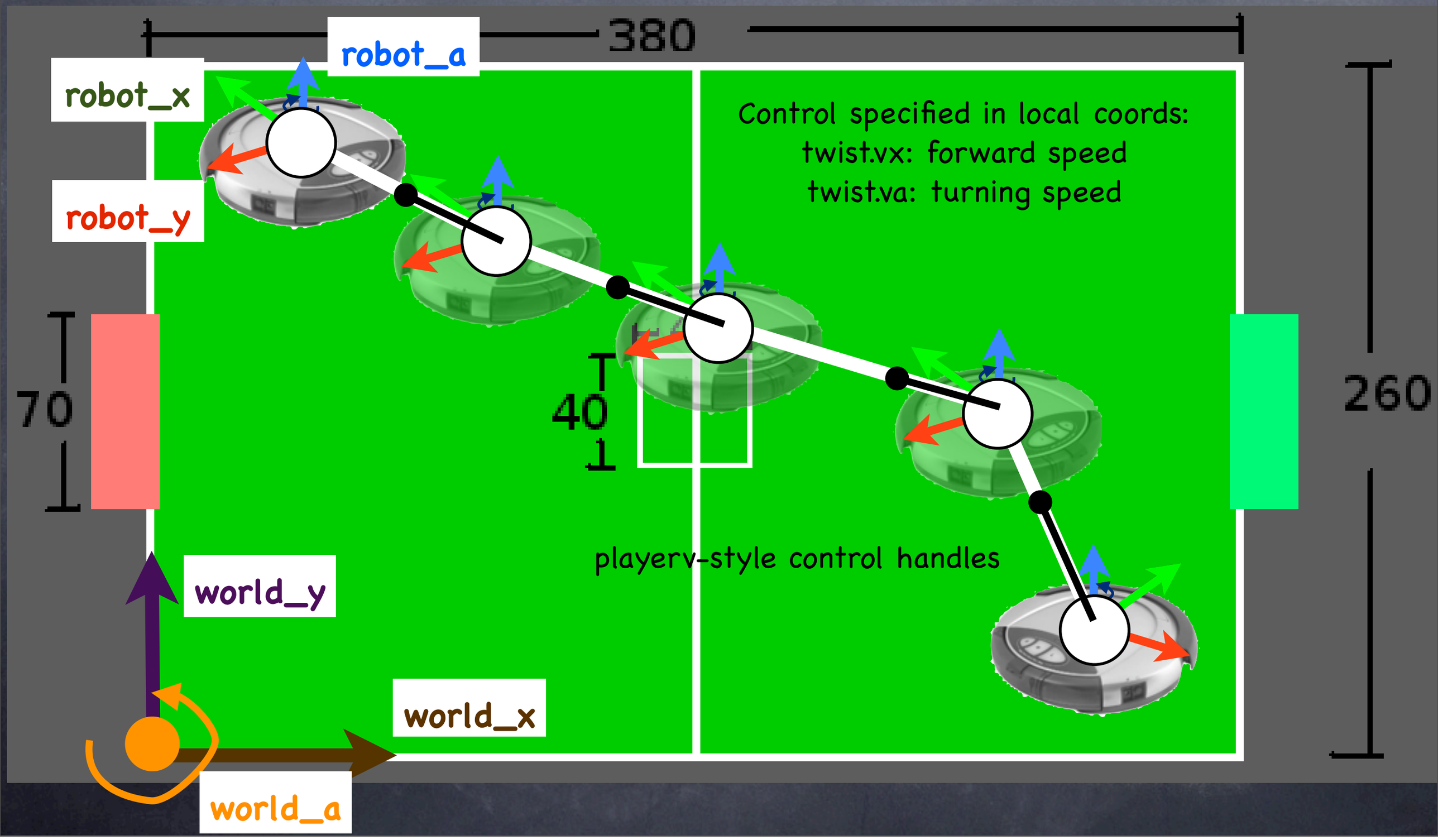
and controls?



1) path planning: sequence of intermediate poses

2) Define pose (3DOF position/orientation on field)

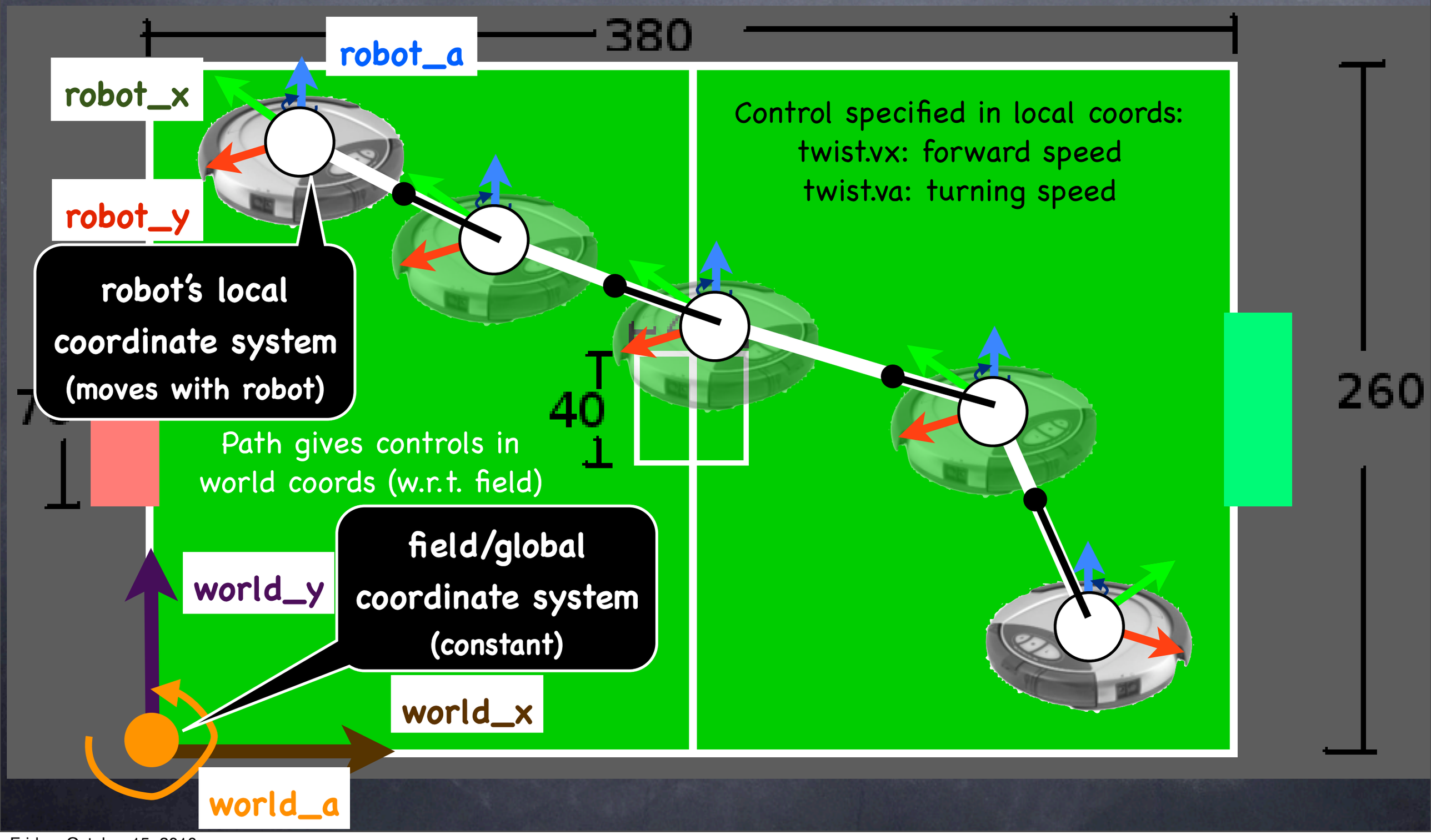
and controls?

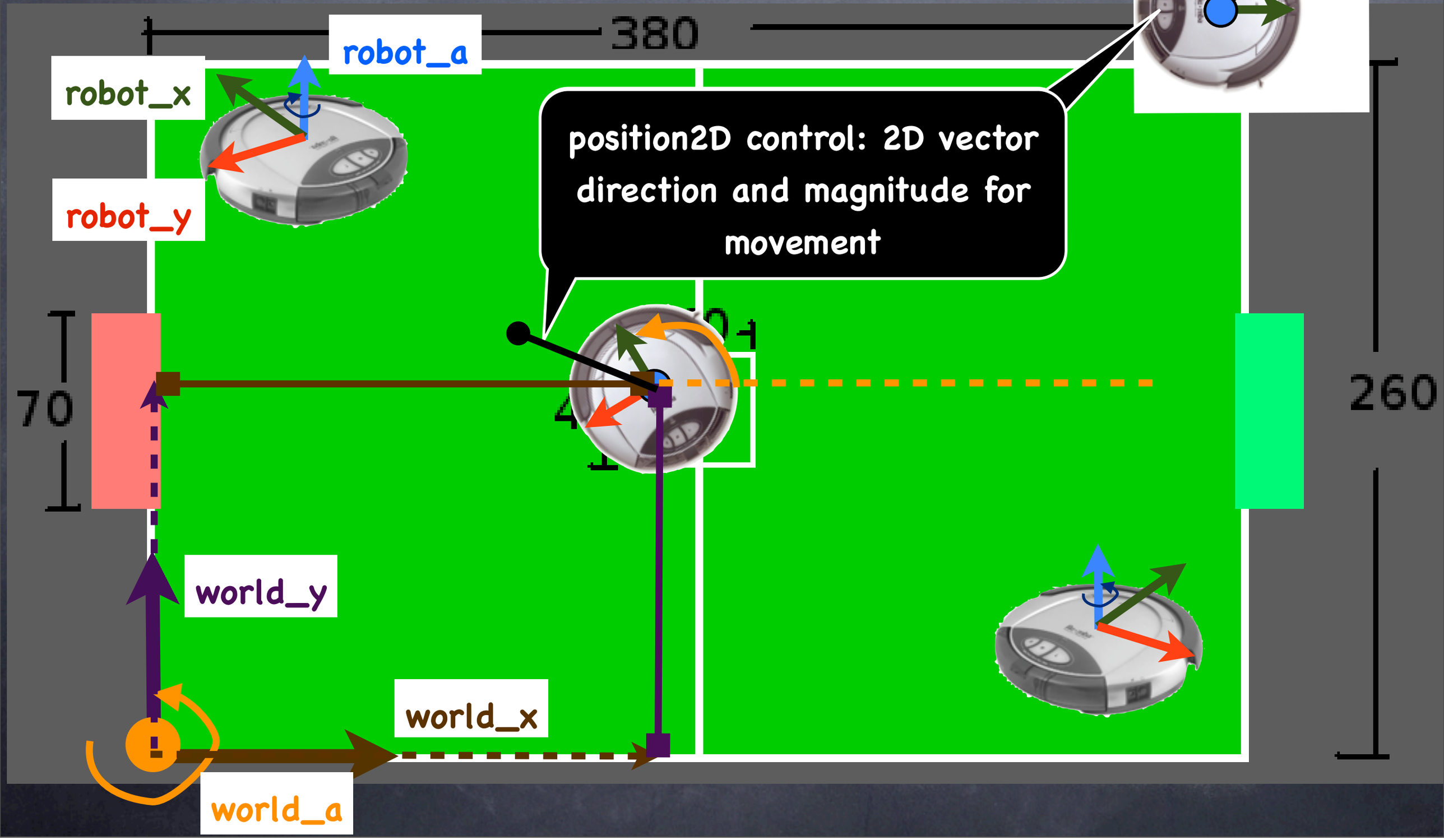


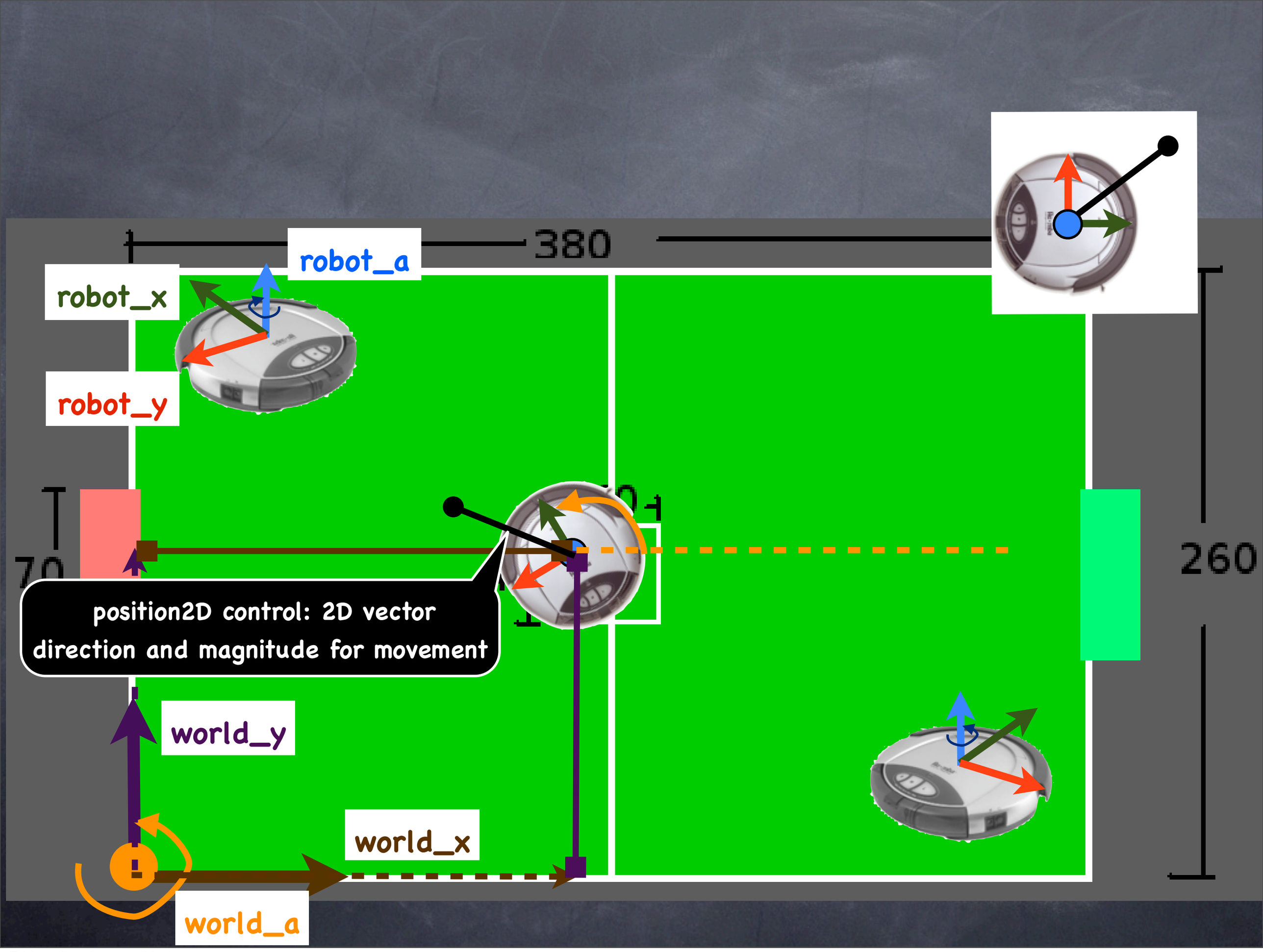
1) path planning: sequence of intermediate poses

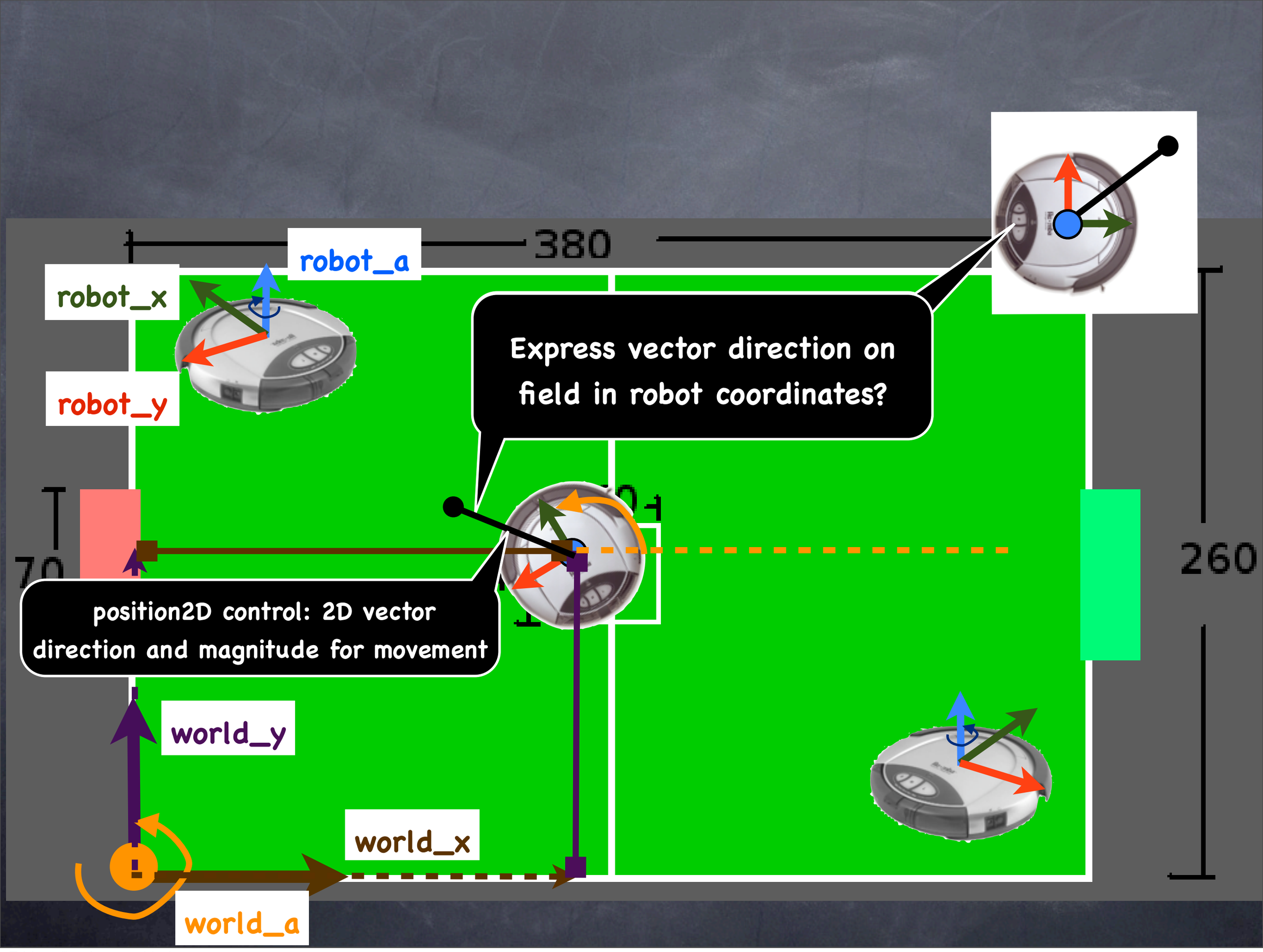
2) Define pose (3DOF position/orientation on field)

and controls (transform global control into robot coordinates)?

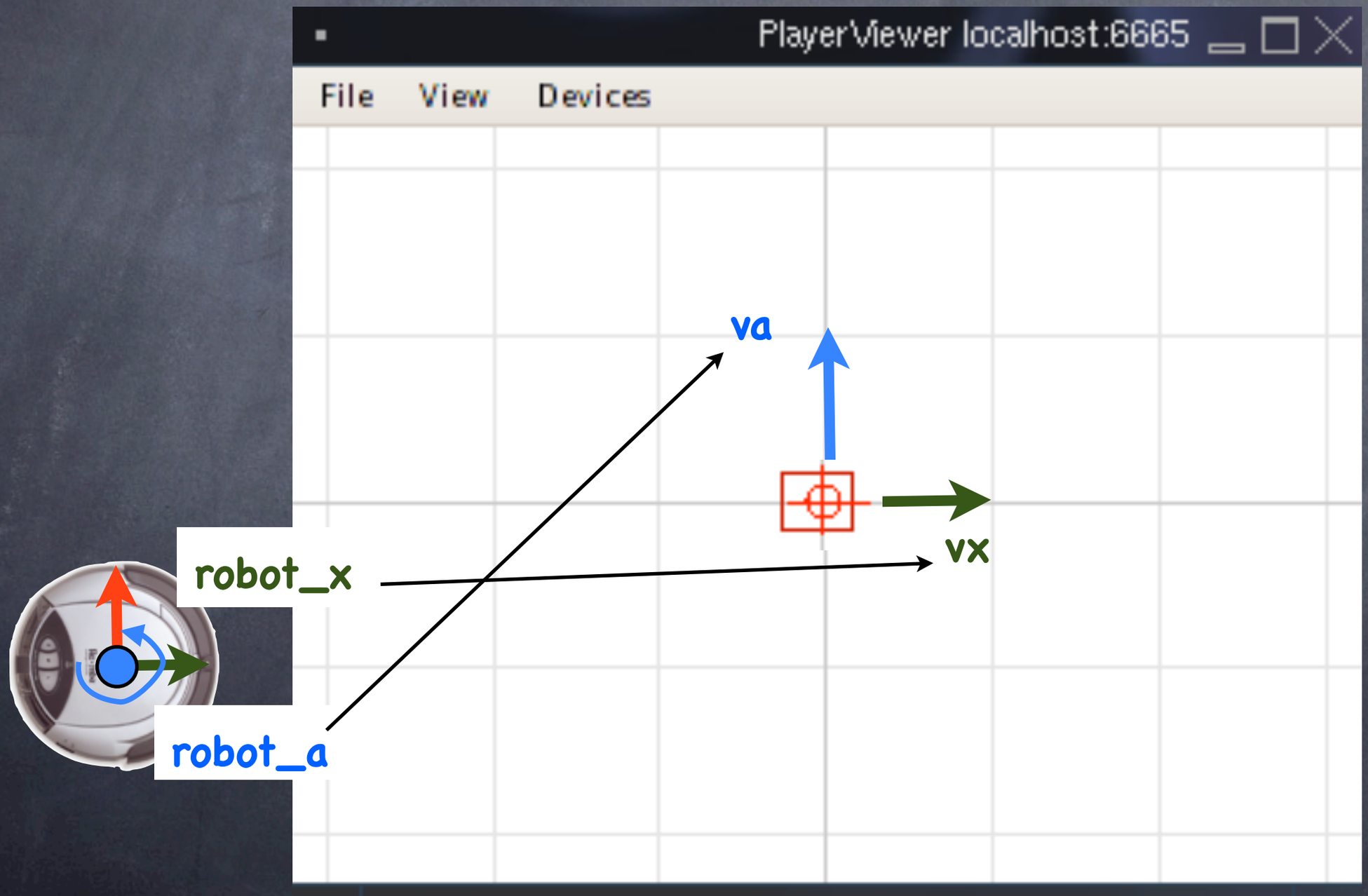



380
robot_a
robot_x
robot_y
position2D control: 2D vector direction and magnitude for movement
70
260
world_y
world_x
world_a

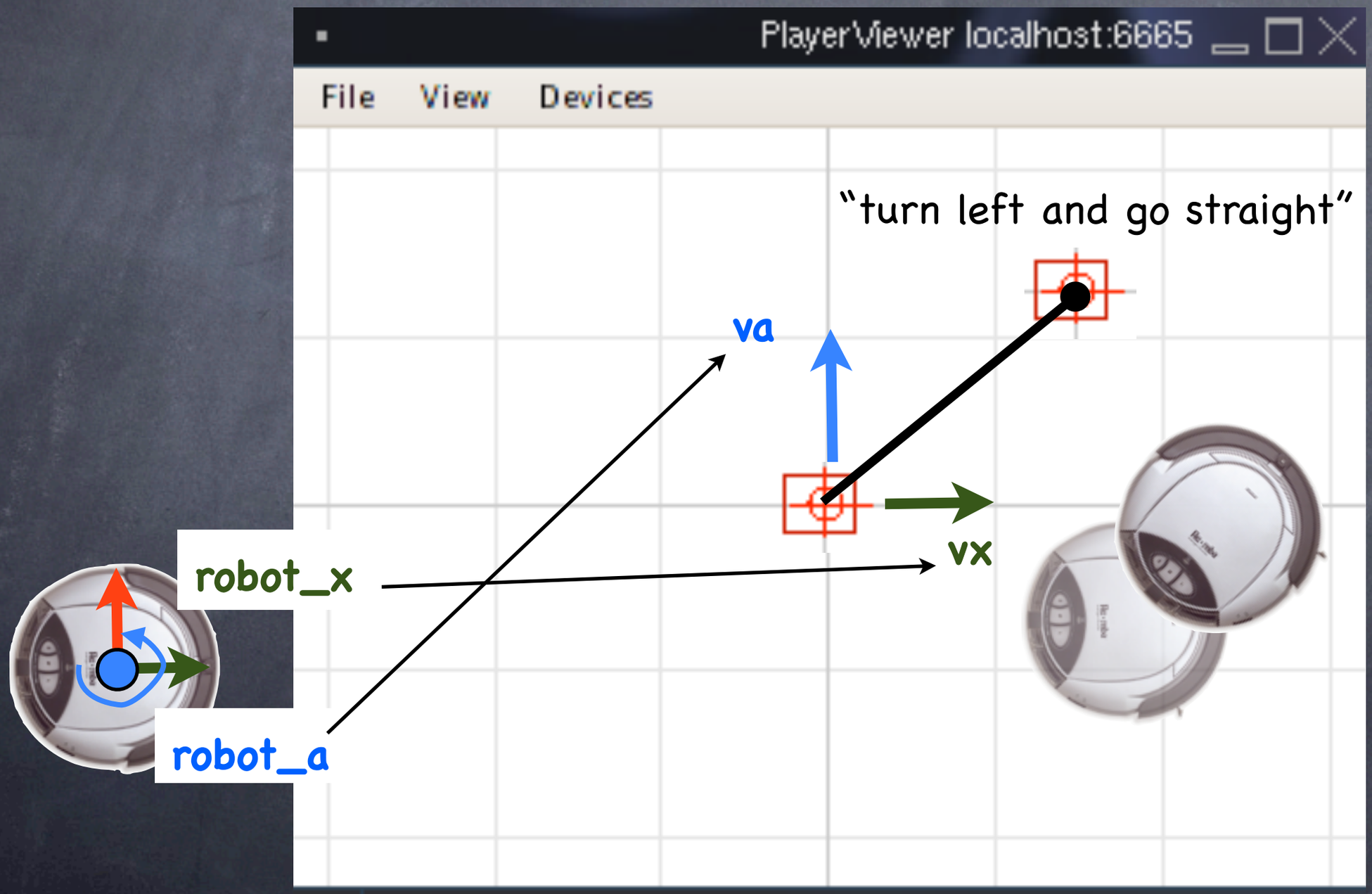

380
robot_a
robot_x
robot_y
260
position2D control: 2D vector
direction and magnitude for movement
world_y
world_x
world_a


380
260
robot_a
robot_x
robot_y
Express vector direction on field in robot coordinates?
position2D control: 2D vector direction and magnitude for movement
world_y
world_x
world_a

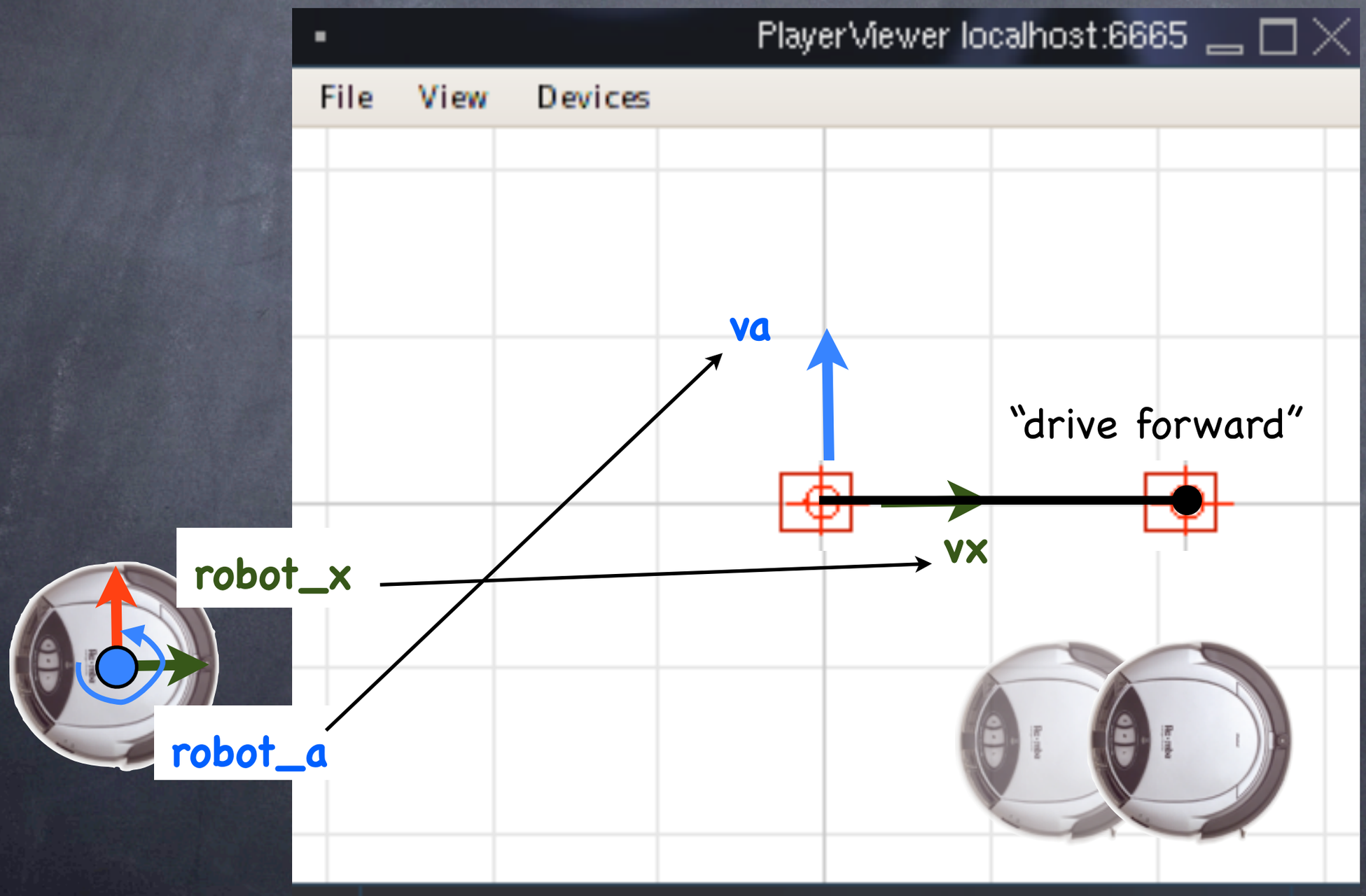
position2d in playerv



position2d in playerv

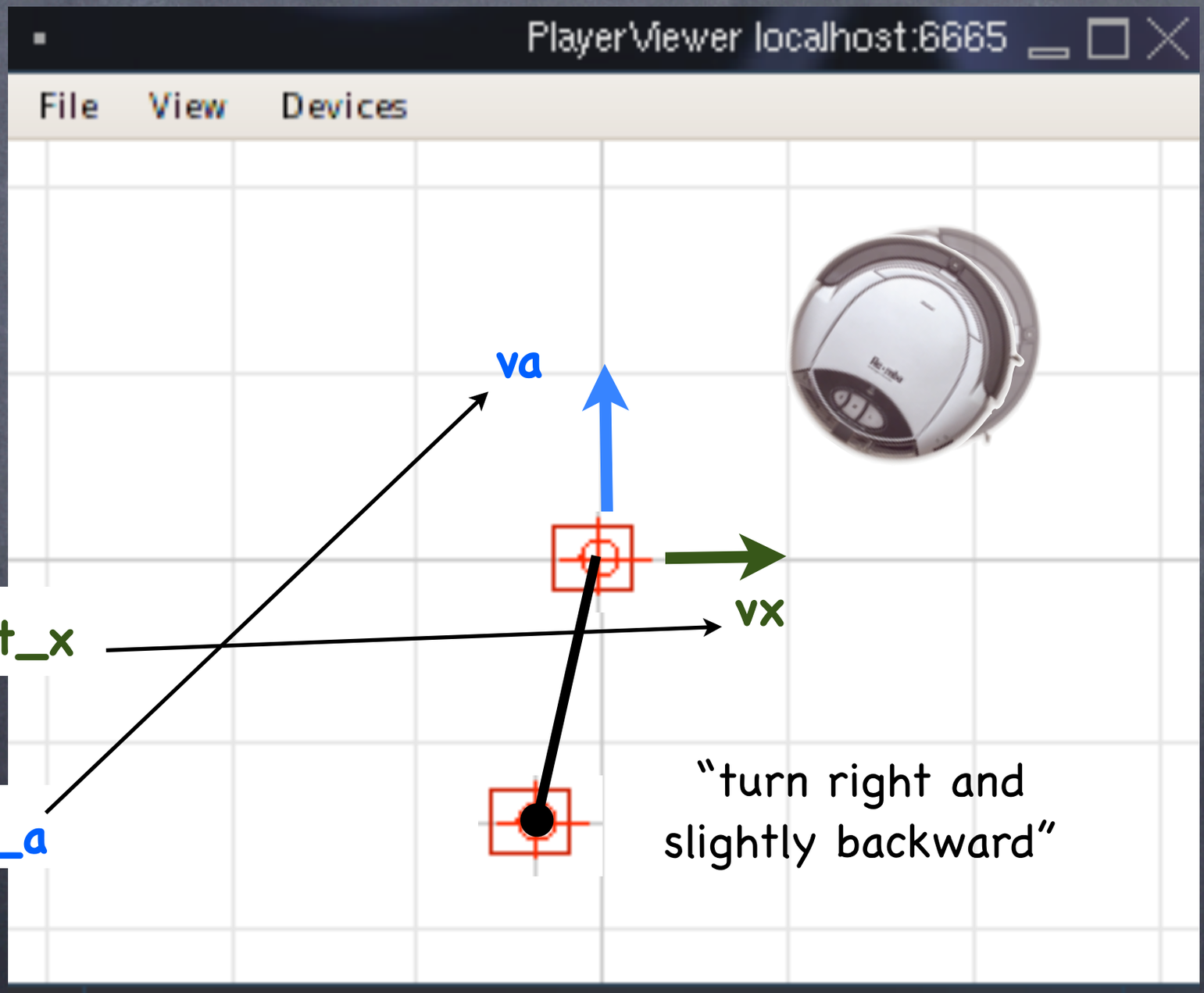


position2d in playerv



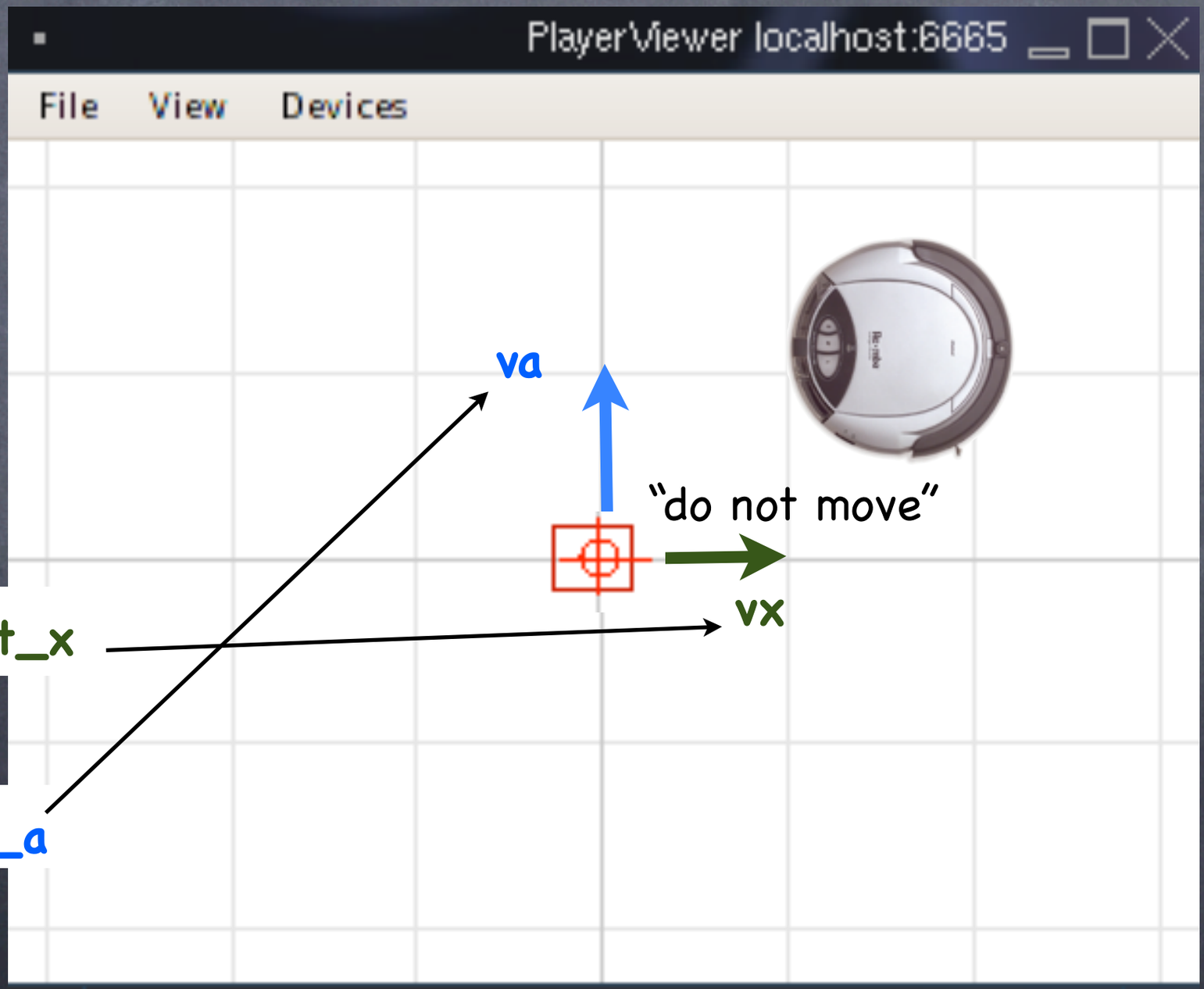
position2d in playerv

Can we be more precise?



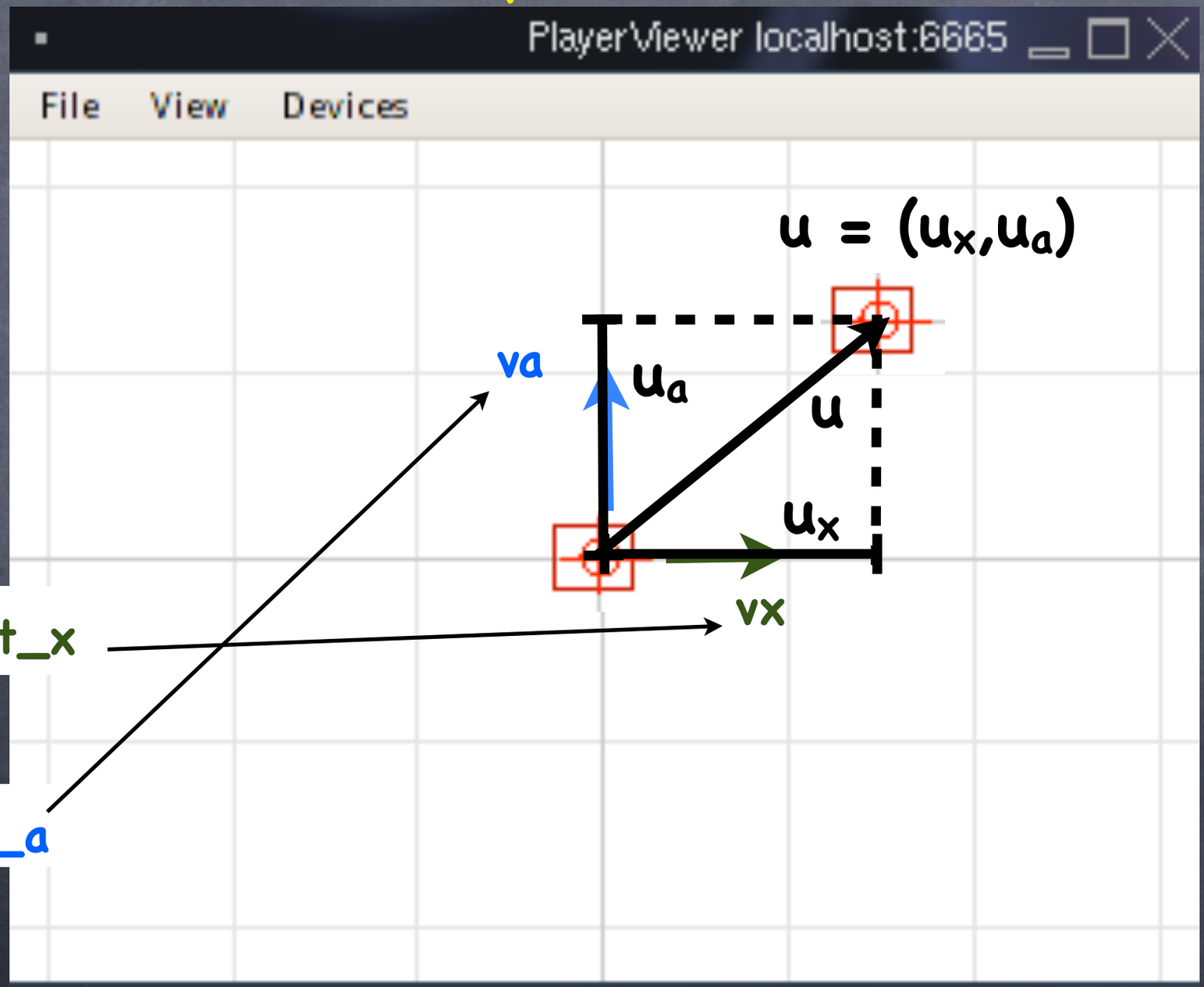
position2d in playerv

Can we be more precise?



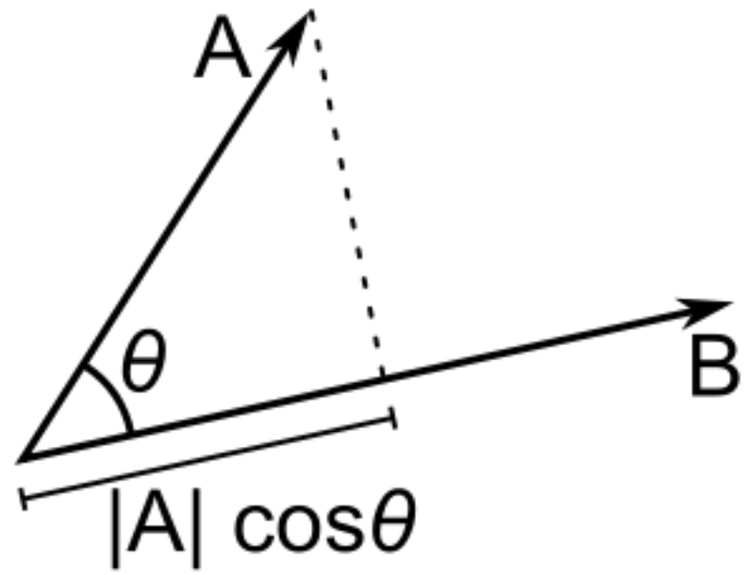
position2d in playerv

Note: u_a is dot product of u and va



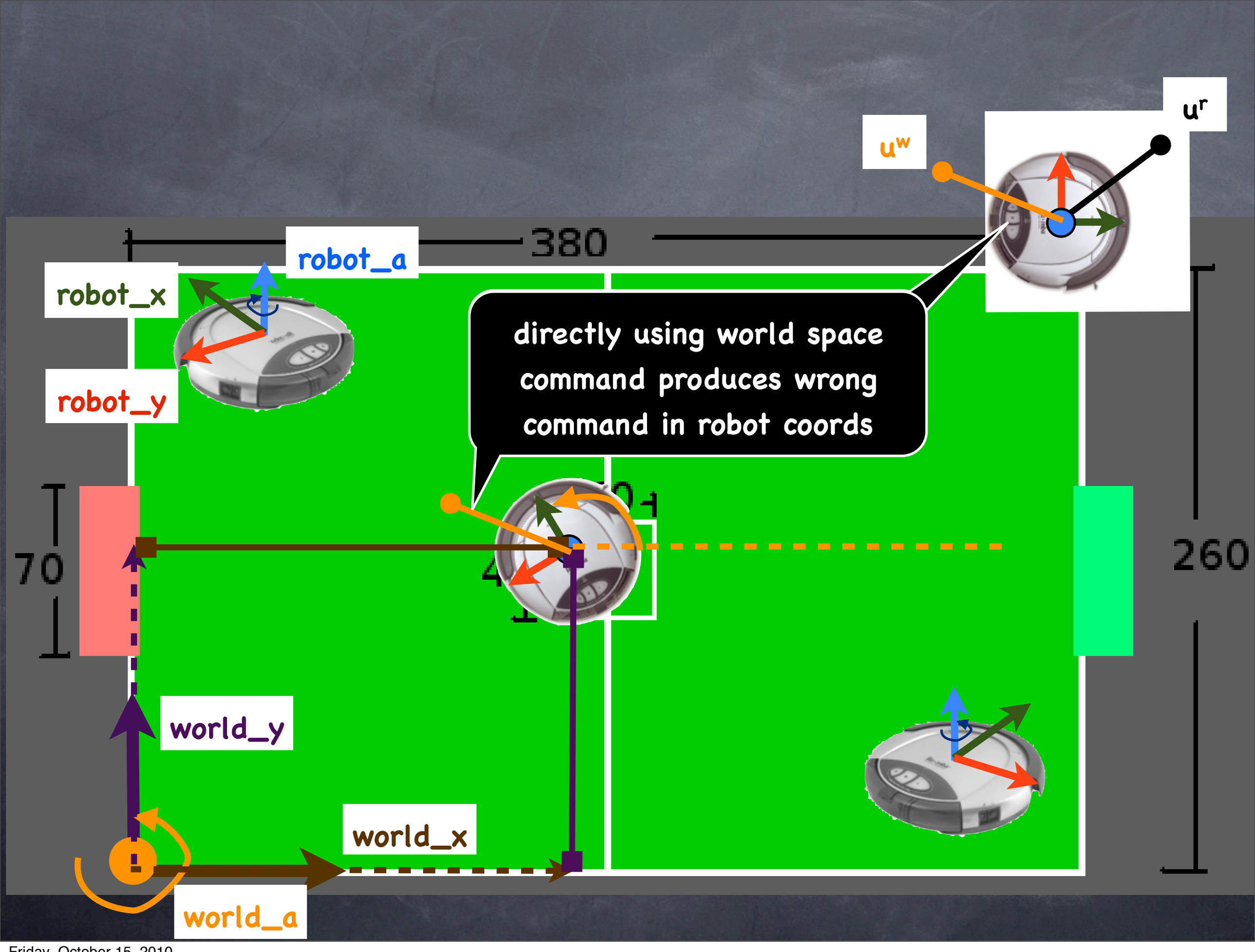
Dot Product

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$



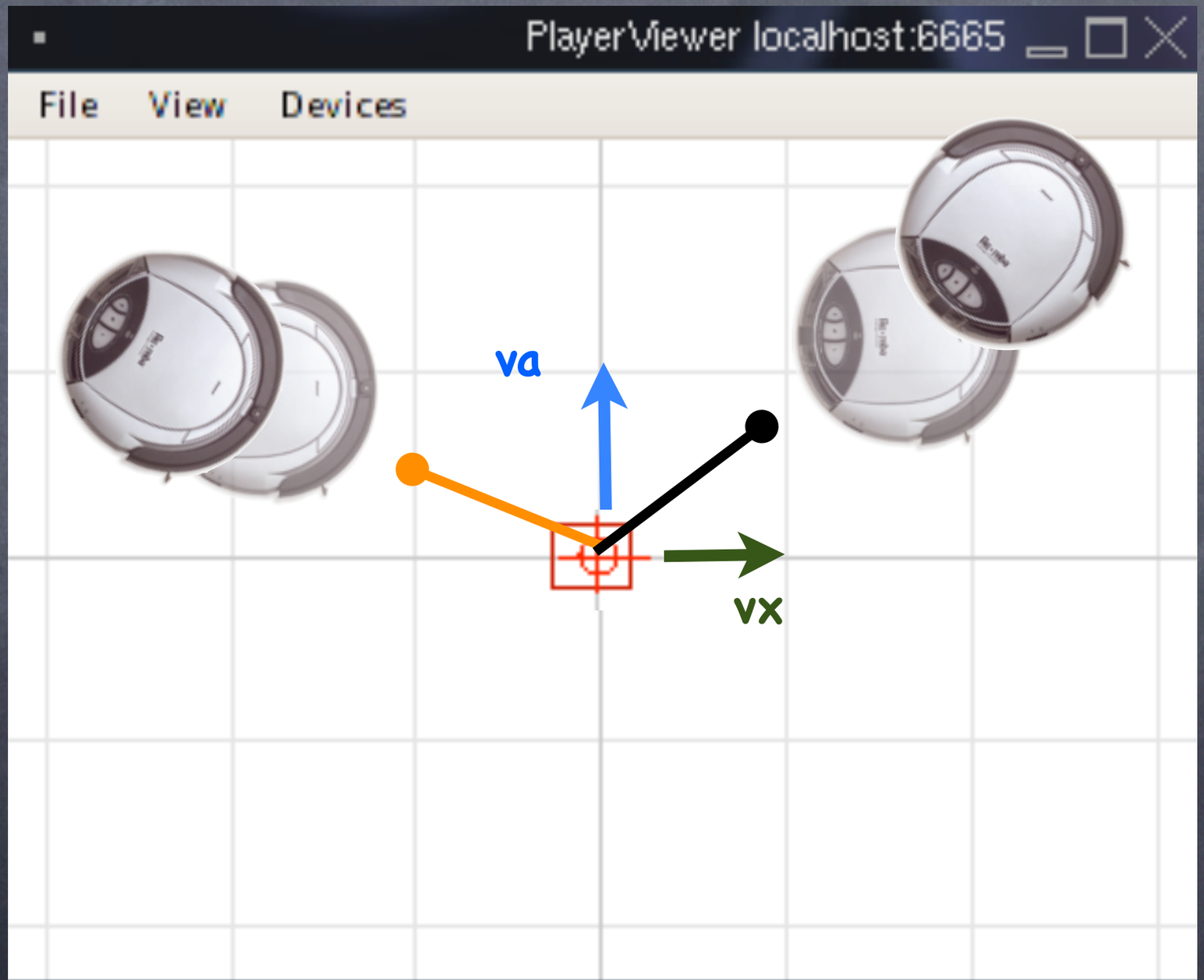
- If one is a unit vector (length 1), the dot product is the projection onto this vector
- Related to angle between vectors

$$\theta = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}\right).$$

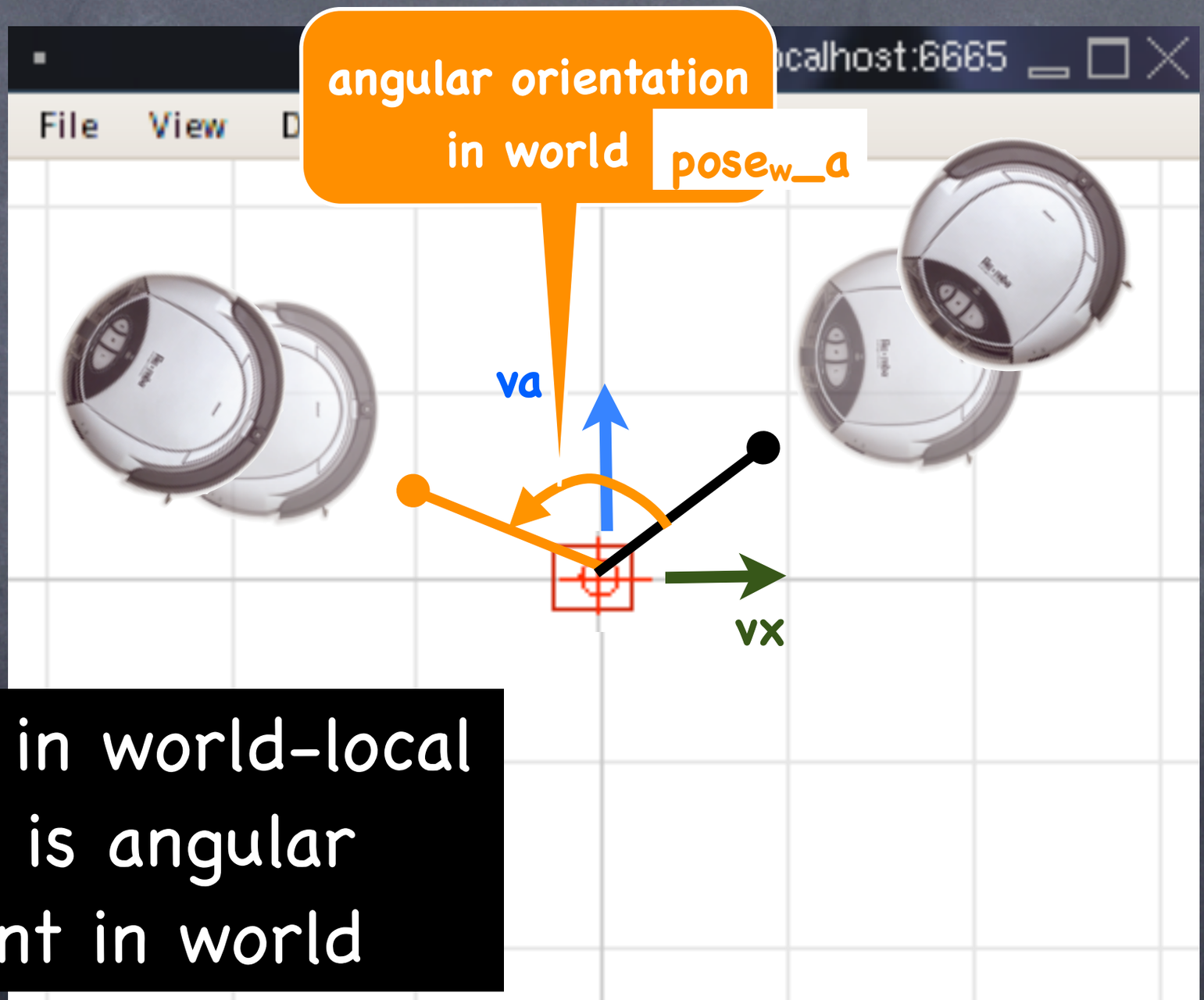

380
robot_a
robot_x
robot_y
directly using world space
command produces wrong
command in robot coords
u^w
u^r
70
260
world_y
world_x
world_a

position2d in playerv

How can we transform control vector into robot coords?

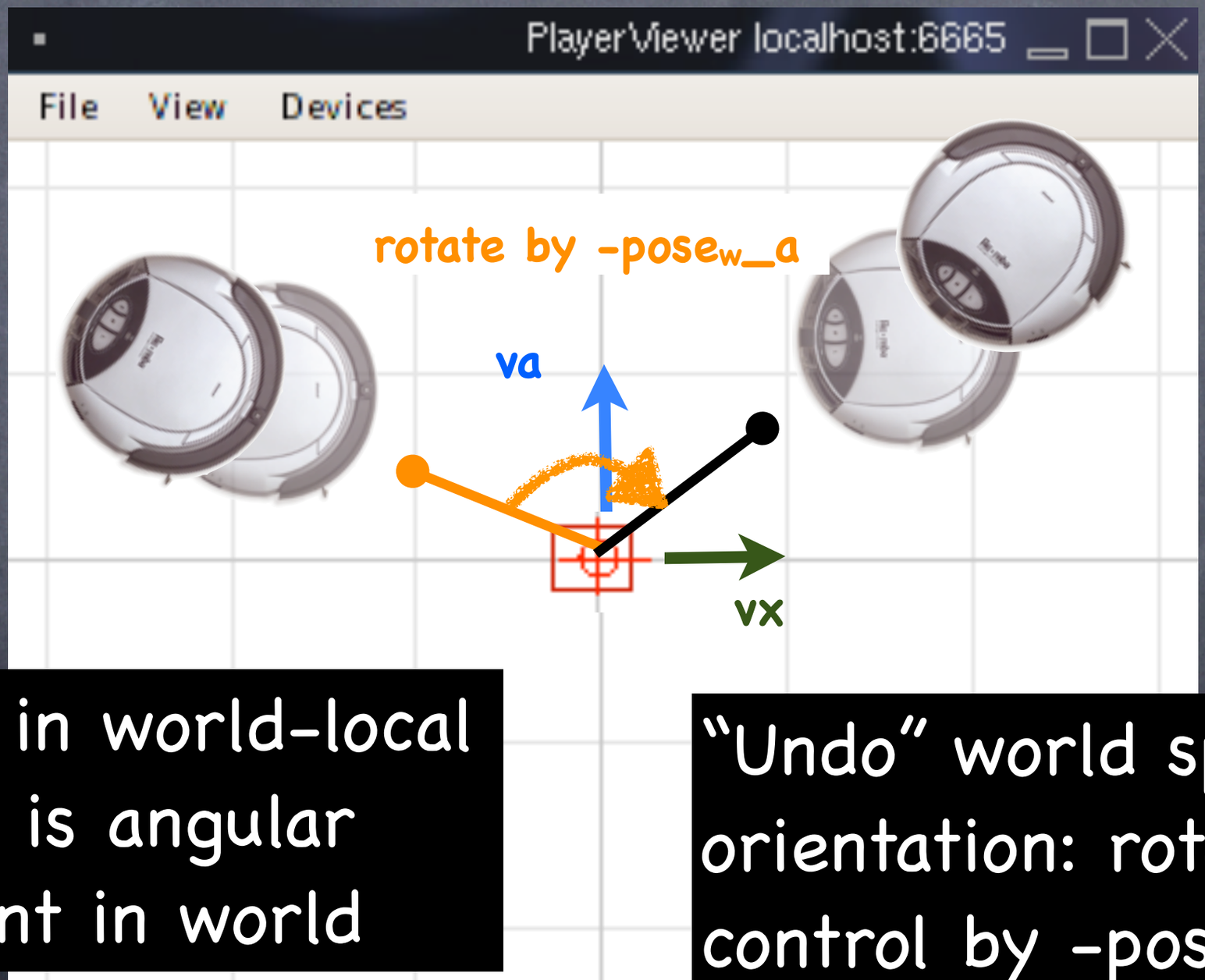


position2d in playerv



Difference in world-local orientation is angular displacement in world

position2d in playerv



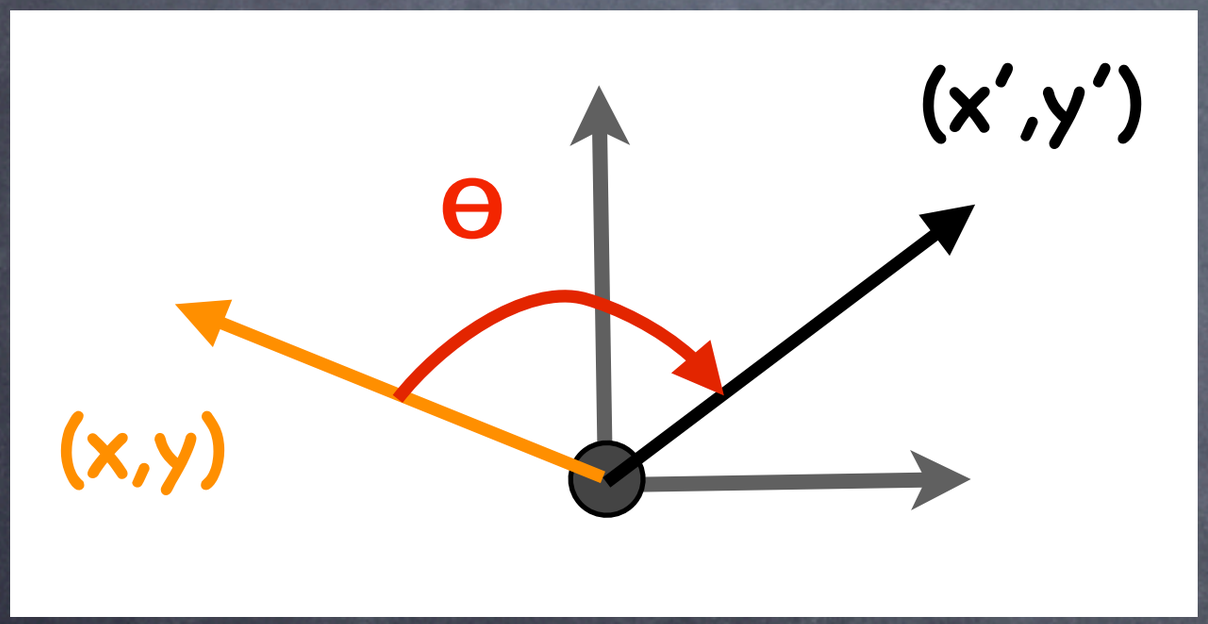
Difference in world-local orientation is angular displacement in world

"Undo" world space orientation: rotate control by -pose$_{w_a}$

Rotating a 2D vector

(counterclockwise)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



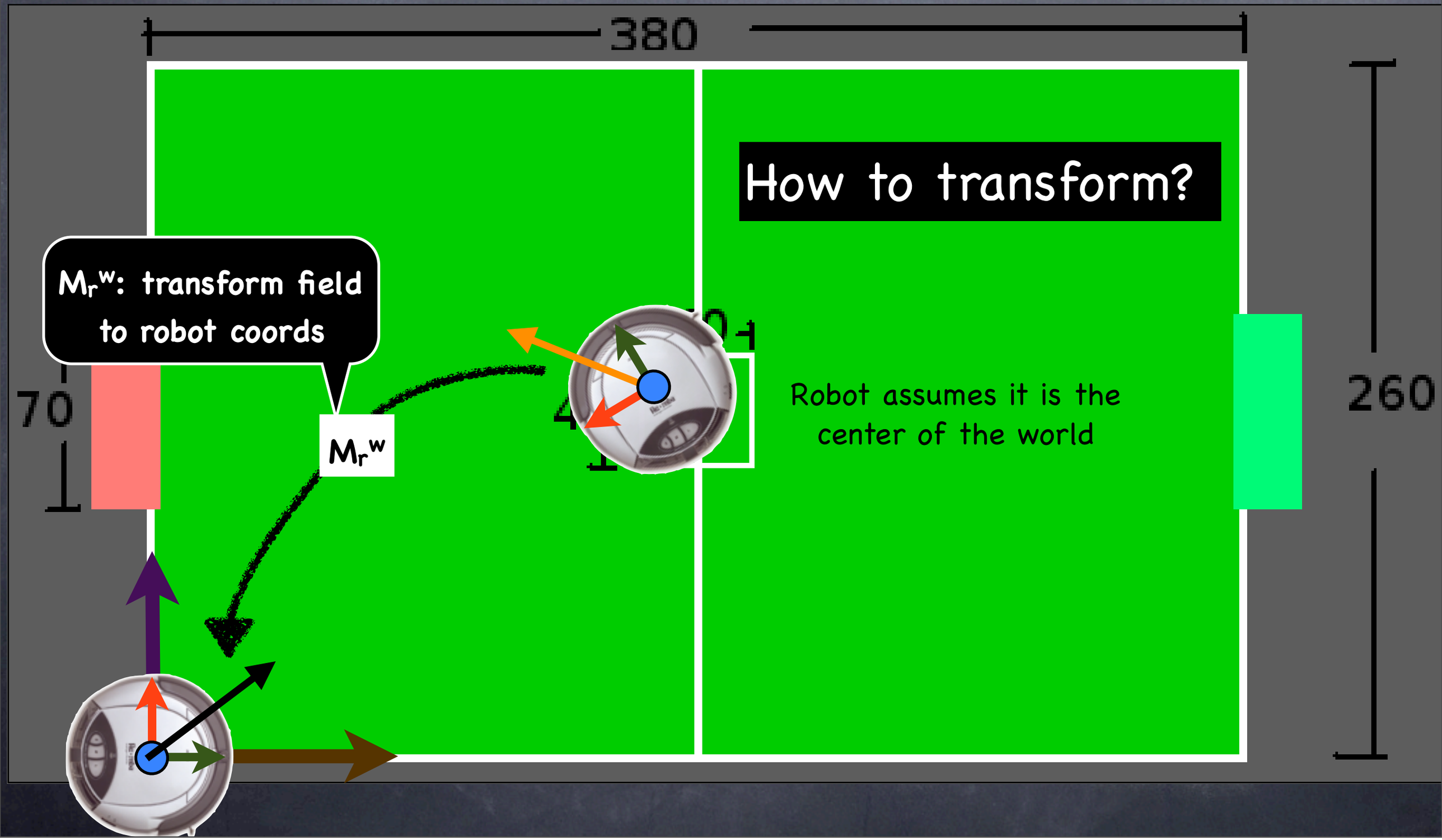
- Matrix multiply vector by 2D rotation matrix
- Matrix parameterized by rotation angle
- Check matrix correctness yourself

Matrix multiplication

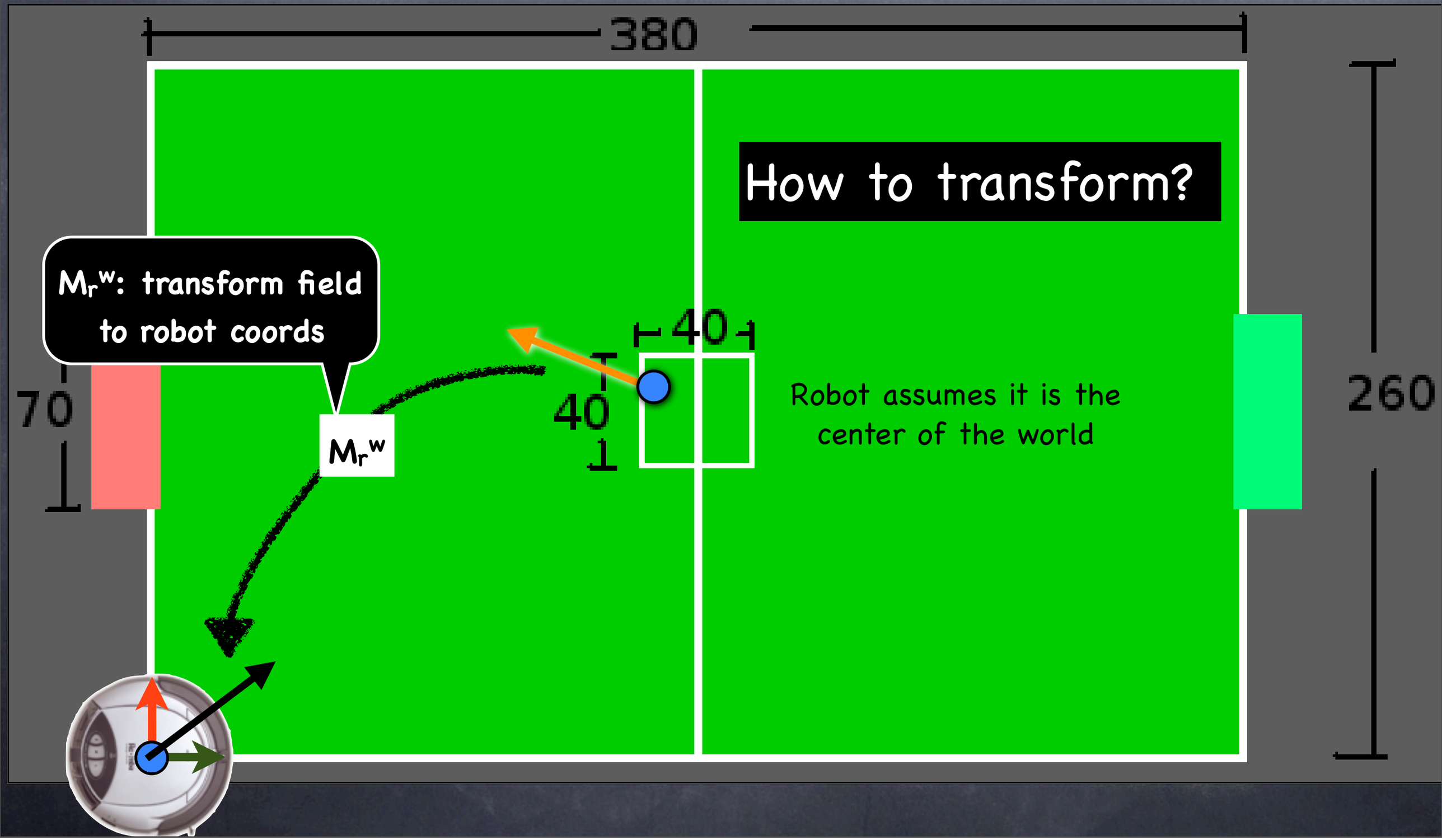
$$\underbrace{\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 1 & 2 & 3 & 4 \end{bmatrix}}_{3\times 4\text{ matrix}} \underbrace{\begin{bmatrix} \cdot & \cdot & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & b & \cdot \\ \cdot & \cdot & \cdot & c & \cdot \\ \cdot & \cdot & \cdot & d & \cdot \end{bmatrix}}_{4\times 5\text{ matrix}} = \underbrace{\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & x_{3,4} & \cdot \end{bmatrix}}_{3\times 5\text{ matrix}}$$

$$x_{3,4} = (1,2,3,4)\cdot(a,b,c,d)$$
$$= 1\times a + 2\times b + 3\times c + 4\times d$$

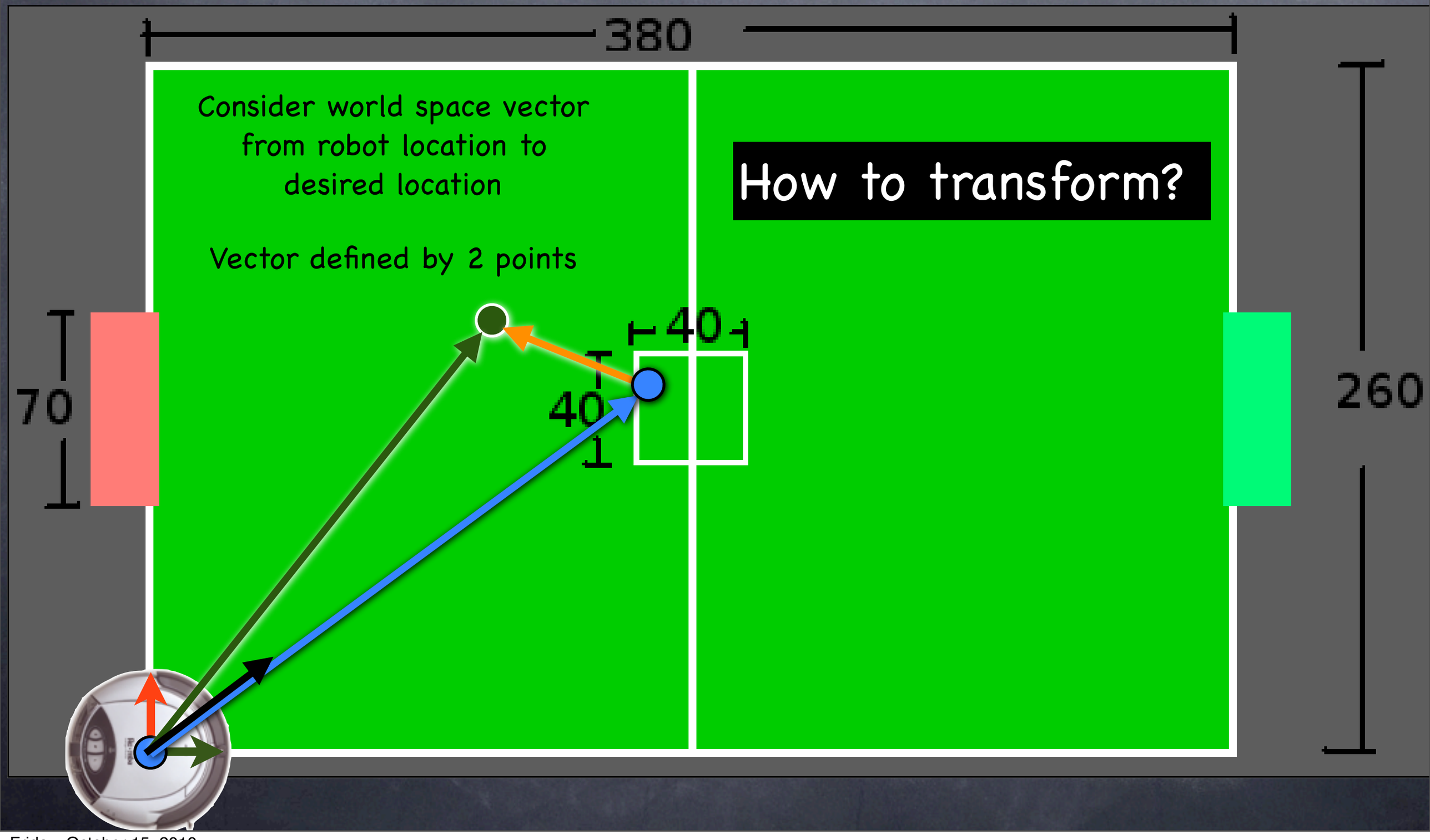
More generally, we should transform world space vector into robot local coordinates to yield command vector



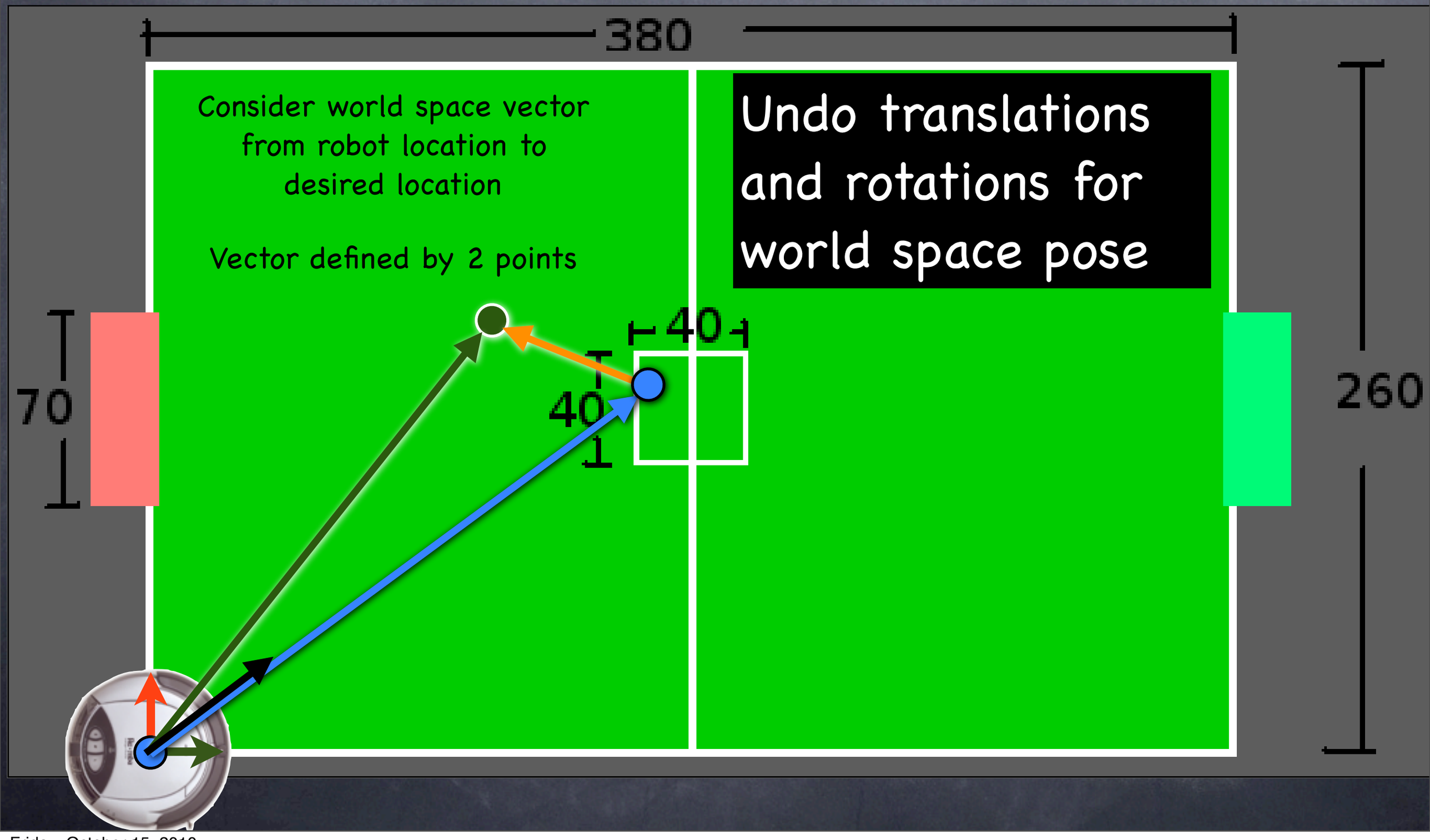
More generally, we should transform world space vector into robot local coordinates to yield command vector



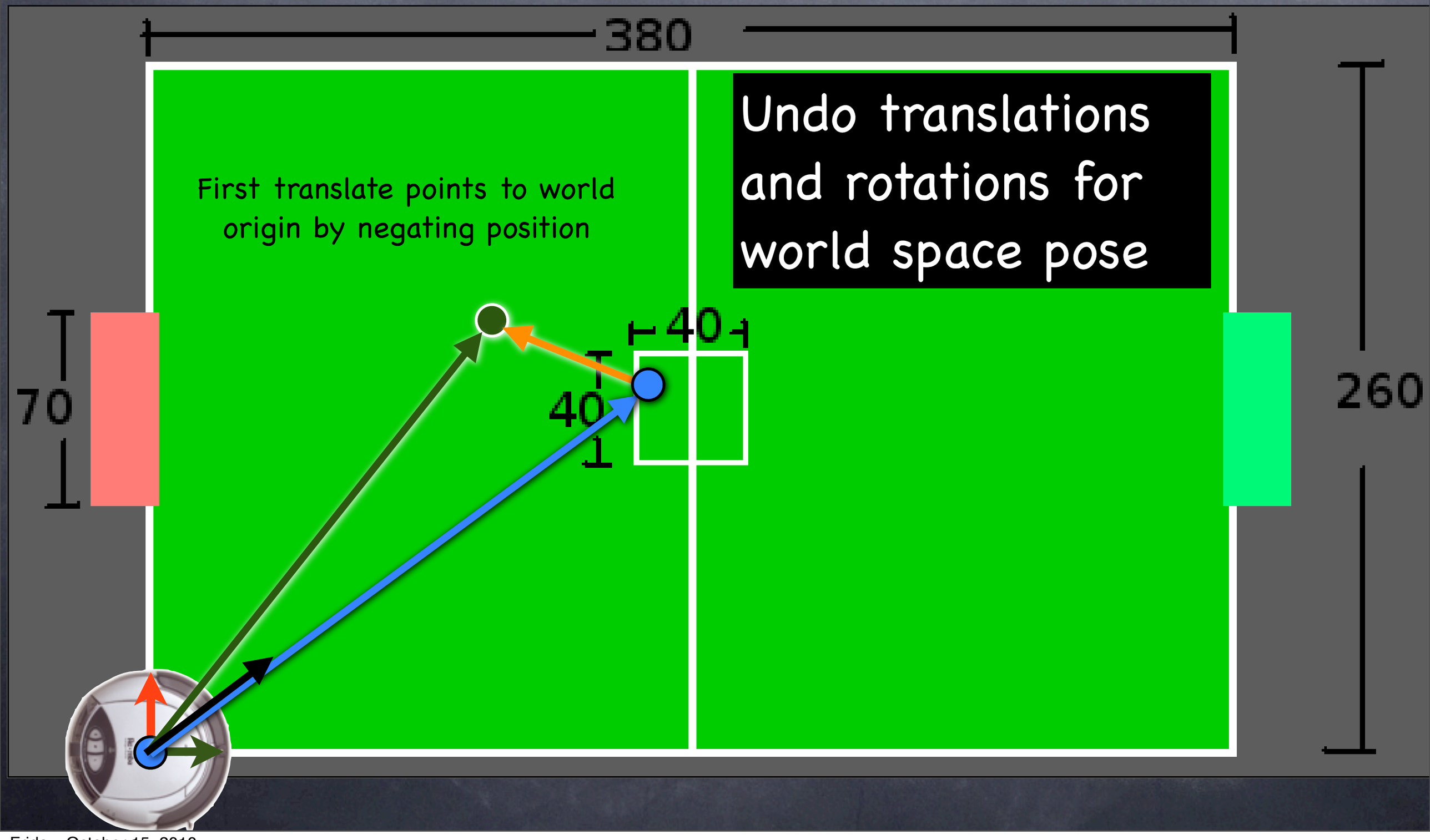
More generally, we should transform world space vector into robot local coordinates to yield command vector

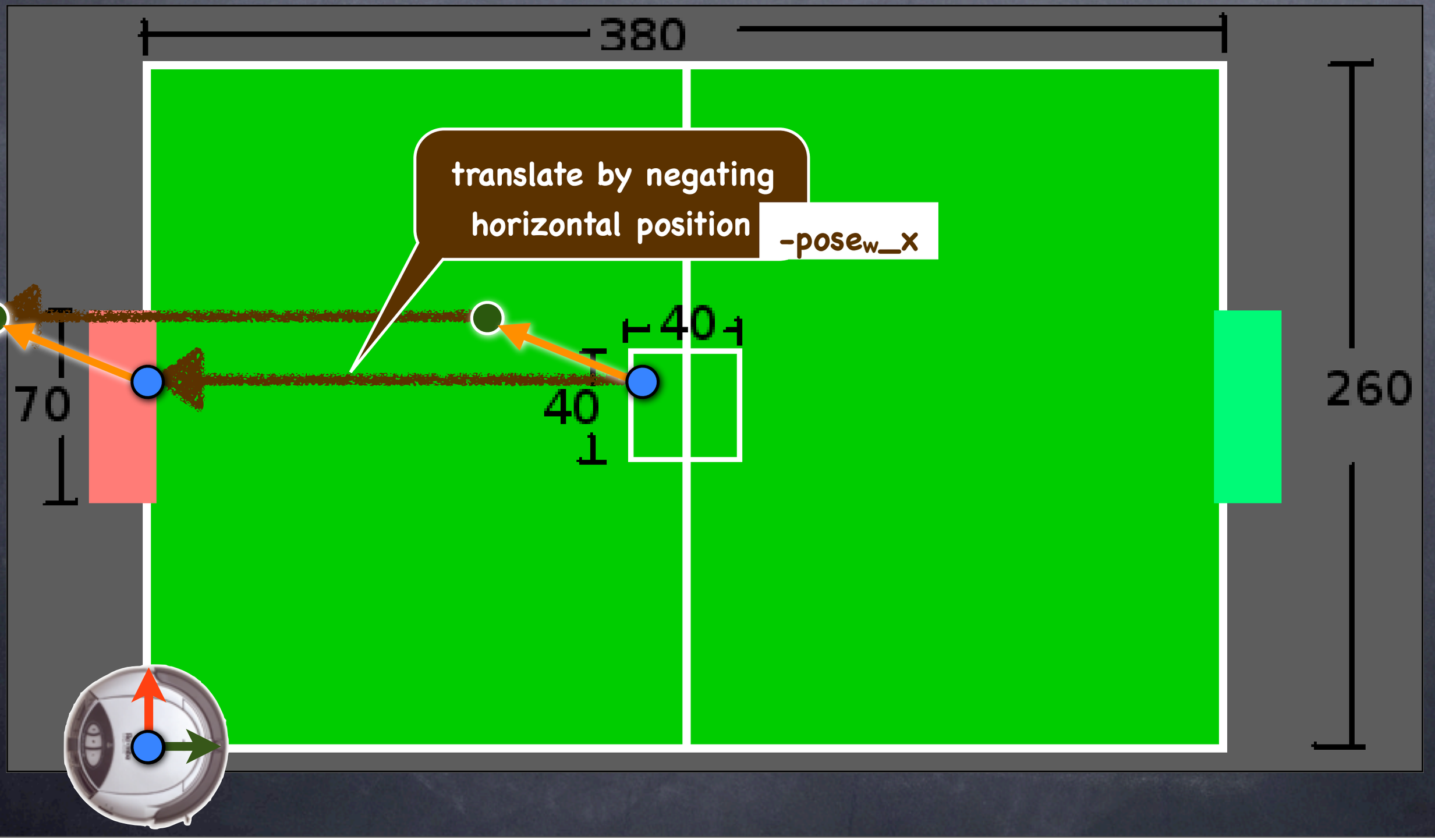


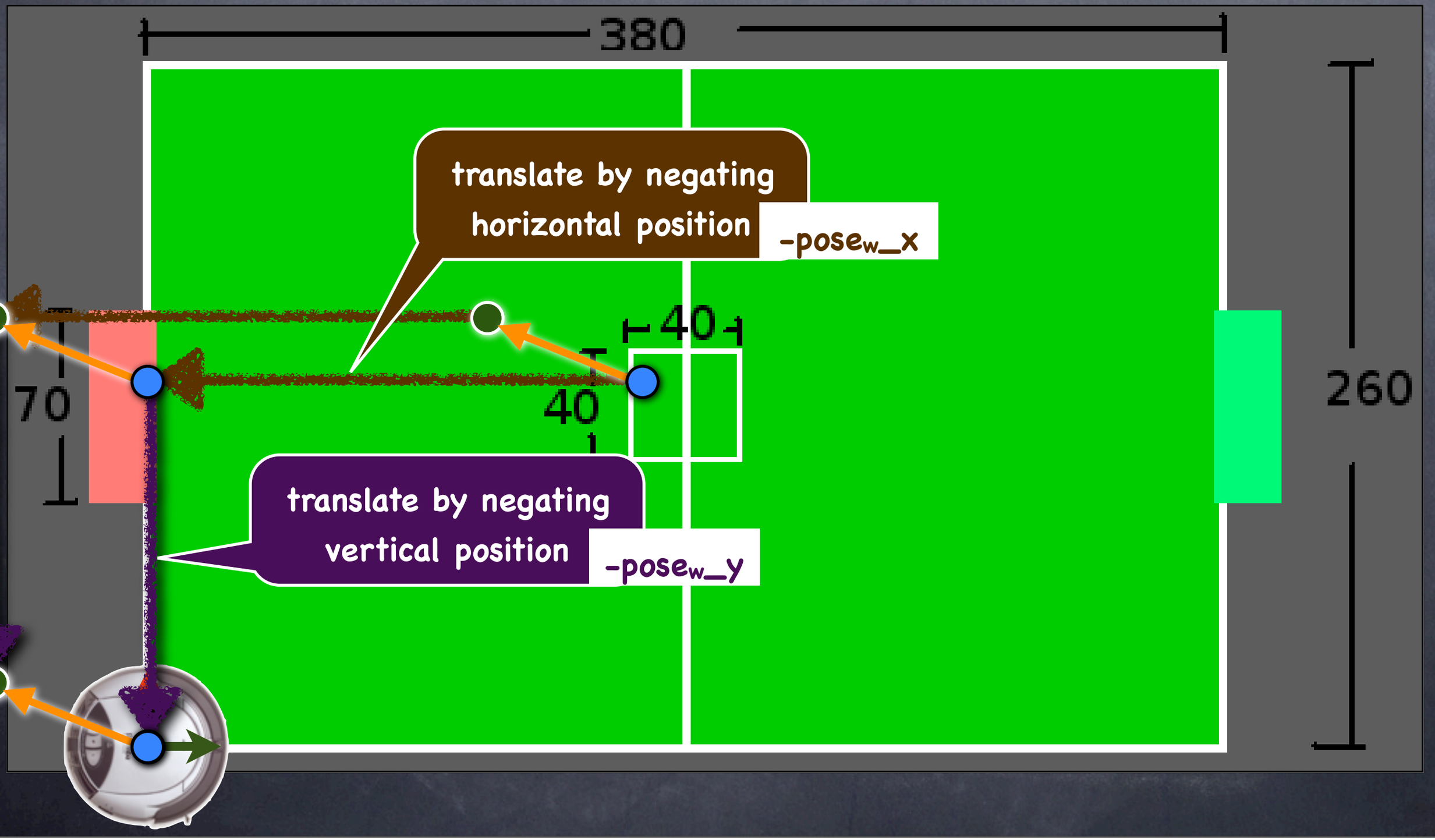
More generally, we should transform world space vector into robot local coordinates to yield command vector



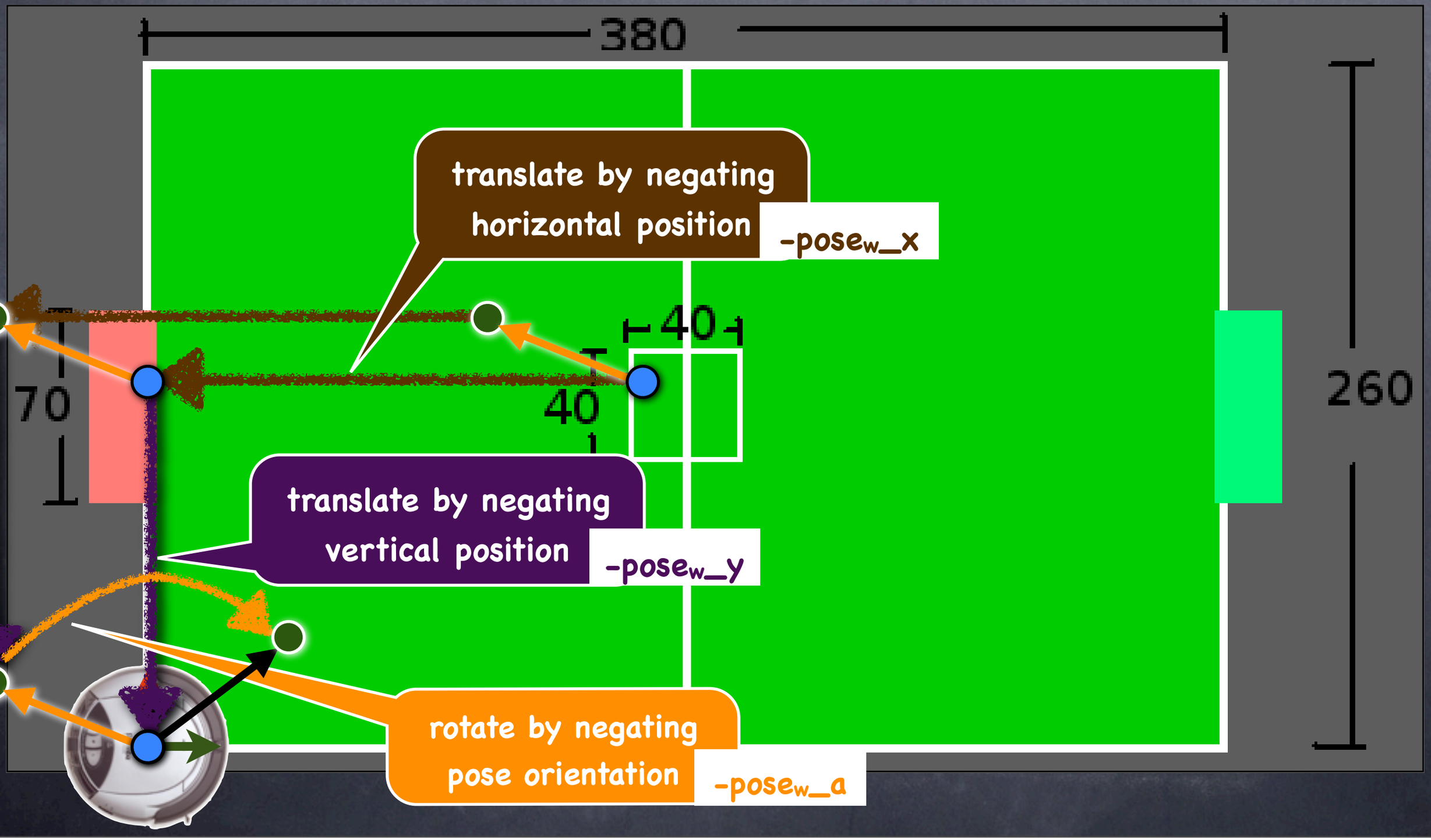
More generally, we should transform world space vector into robot local coordinates to yield command vector




380
translate by negating horizontal position
-pose_w_x
40
40
70
260

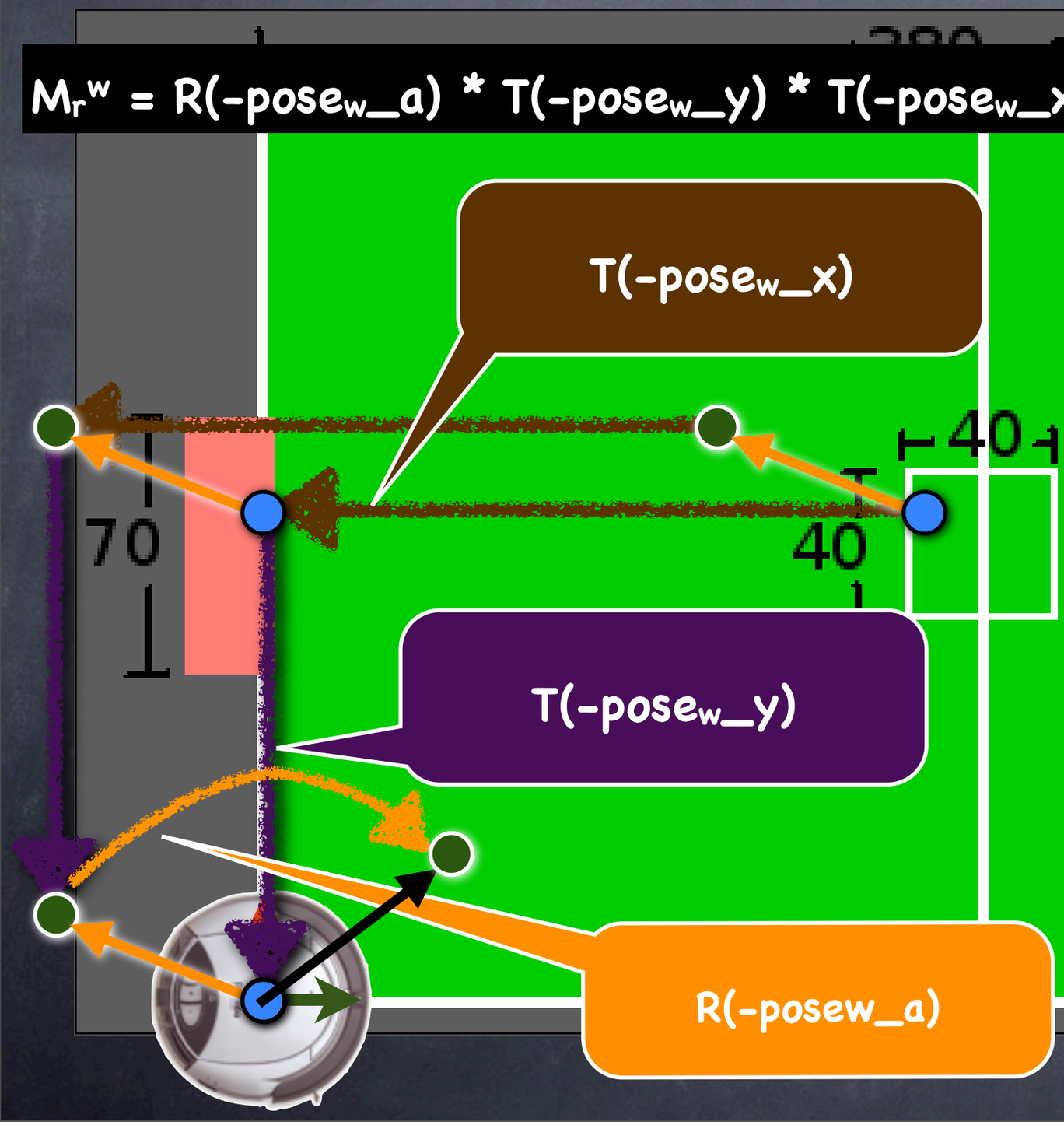

380
translate by negating
horizontal position
-pose_w_x
40
40
70
260
translate by negating
vertical position
-pose_w_y

Pose space vector results from undoing world space translations and rotations on 2 points



Pose space vector results from undoing world space translations and rotations on 2 points

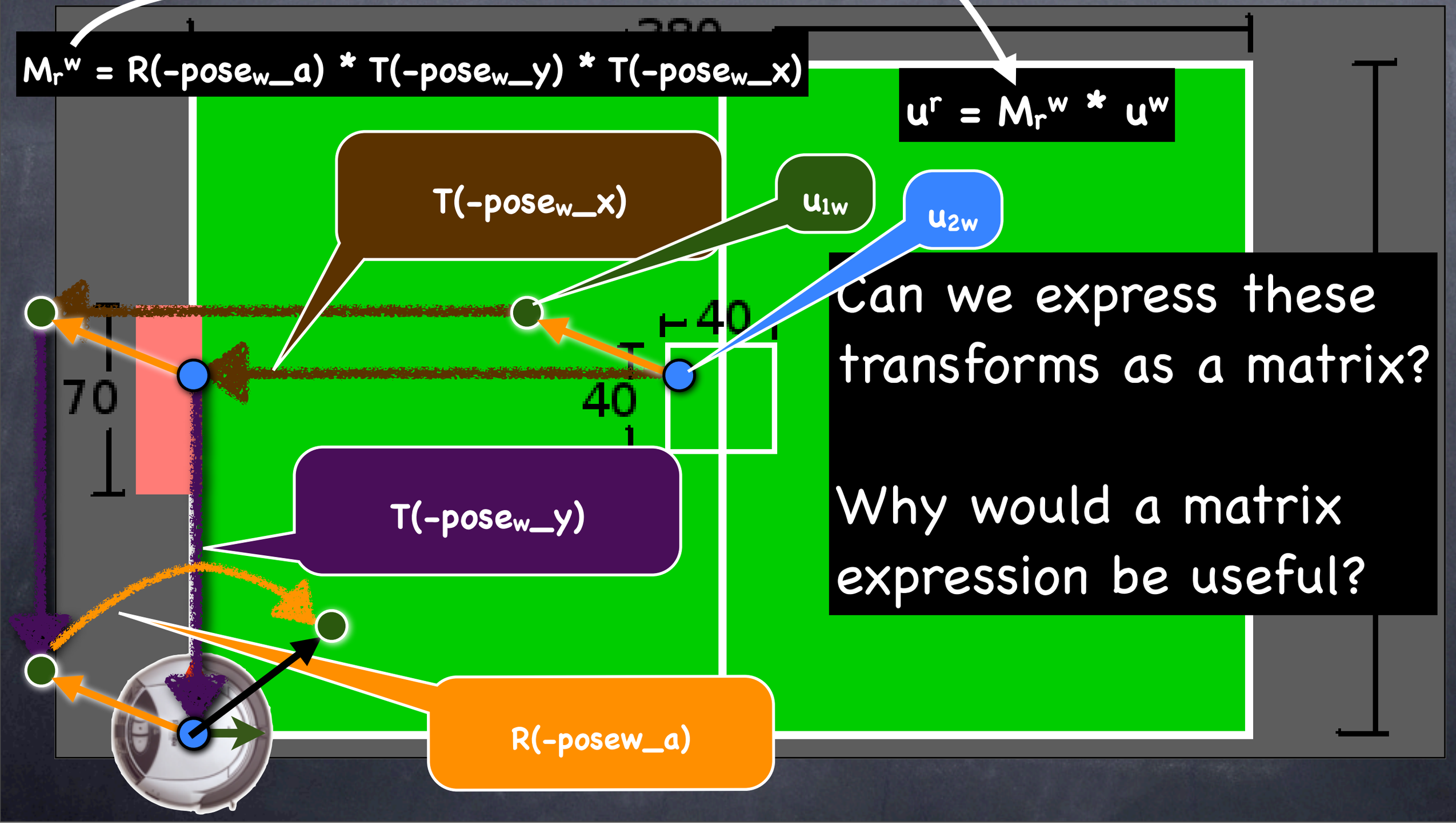
$M_r^w = R(-pose_w\_a) * T(-pose_w\_y) * T(-pose_w\_x)$



Can we express these transforms as a matrix?

Why would a matrix expression be useful?

Transform accomplished by multiplying points of vector by "pose-to-world" matrix



Are we all set to transform our vectors?

$M_r^w = R(-pose_w\_a) * T(-pose_w\_y) * T(-pose_w\_x)$

$u_r = M_r^w * u_w$

$T(-pose_w\_x)$

u_{1w}

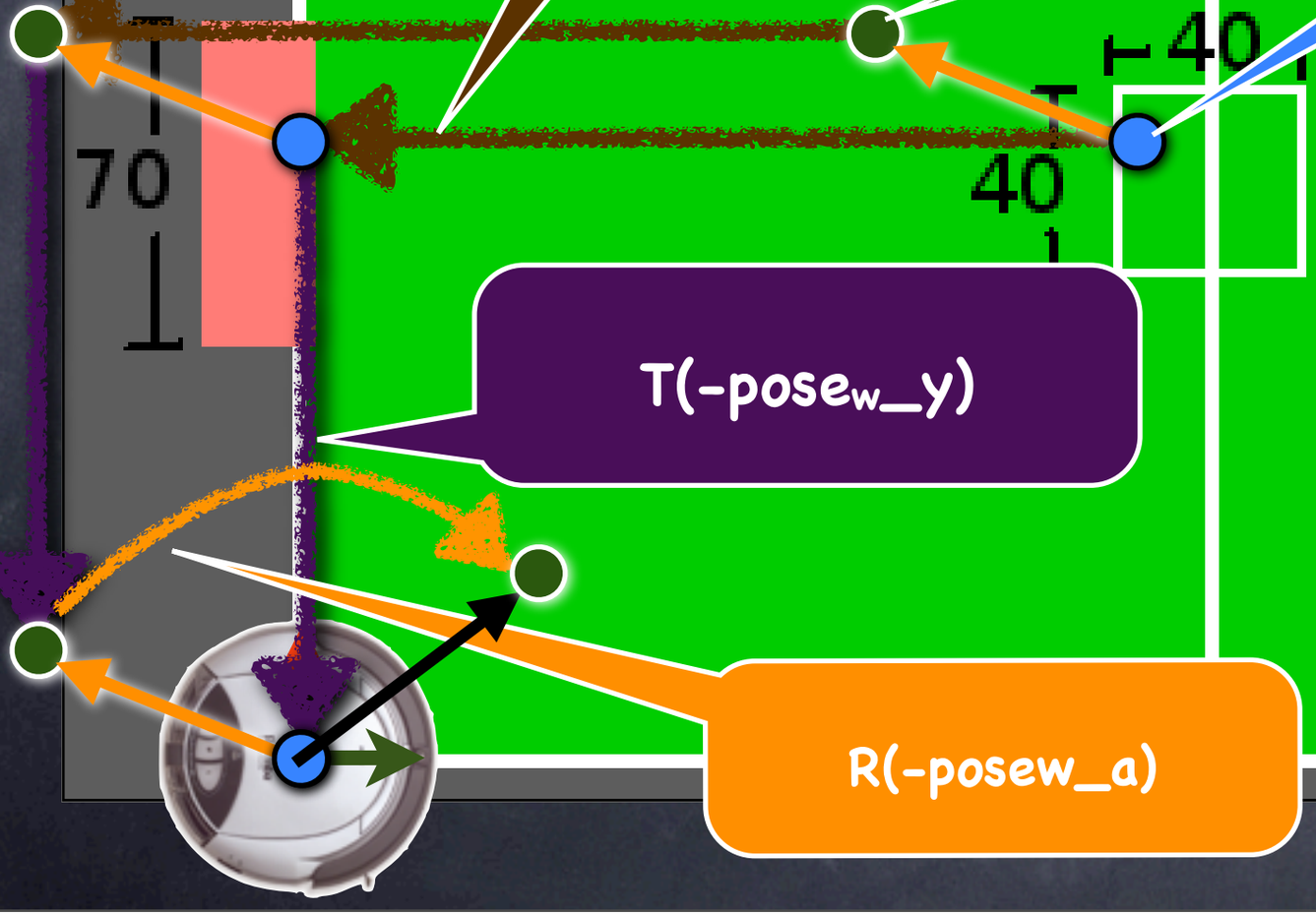
u_{2w} (becomes pose space origin)

Can we express these transforms as a matrix?

Why would a matrix expression be useful?

$T(-pose_w\_y)$

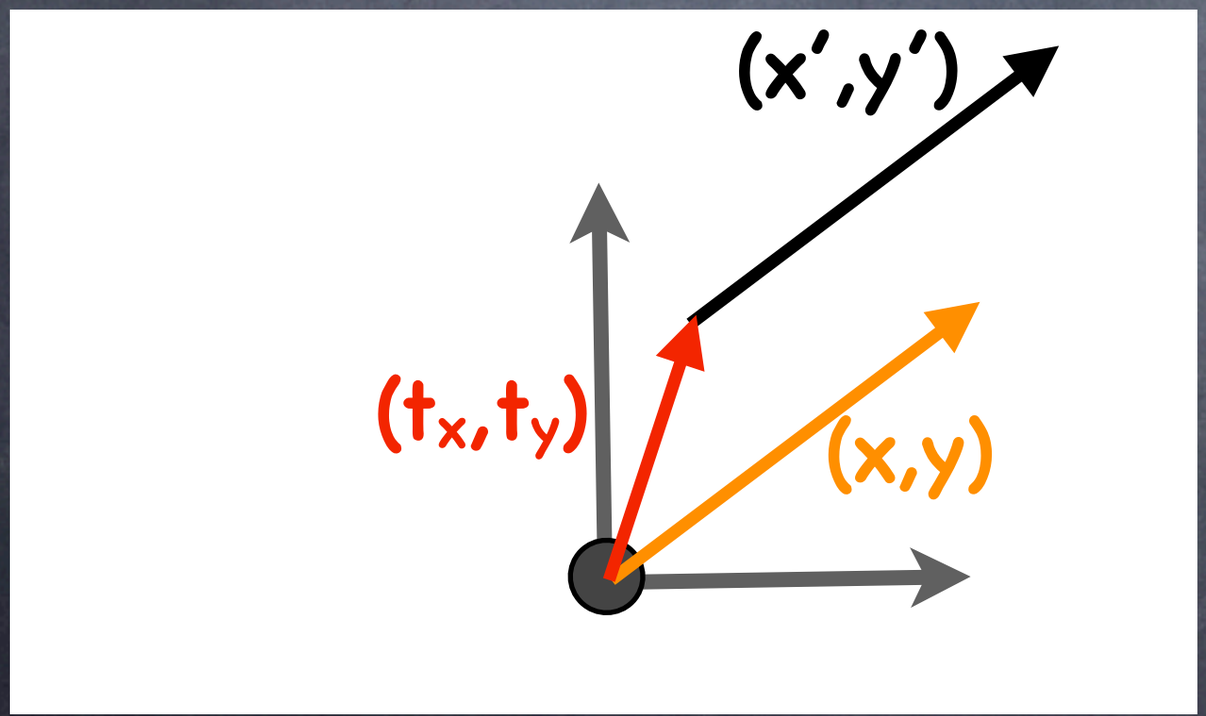
$R(-posew\_a)$



2D Translation

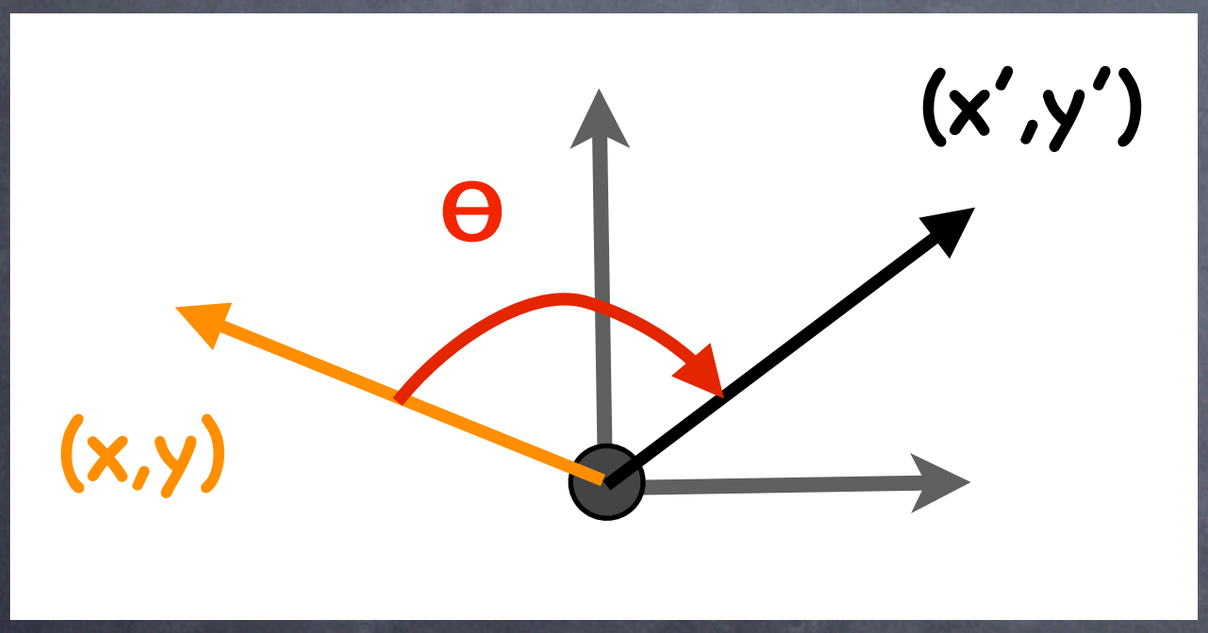
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Requires "homogeneous coordinates"



- 3D vector with position concatenated with a 1
- Matrix parameterized by horizontal and vertical displacement

Homogeneous 2D rotation matrix



$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

u^r | Rotate to align | Translate to origin | u^w

$$\begin{bmatrix} u_{1x}^r & u_{2x}^r \\ u_{1a}^r & u_{2a}^r \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{1x}^w & u_{2x}^w \\ u_{1y}^w & u_{2y}^w \\ 1 & 1 \end{bmatrix}$$

M_w^r

Θ : pose rotation

t_x, t_y: pose position

u^w

M_w^r

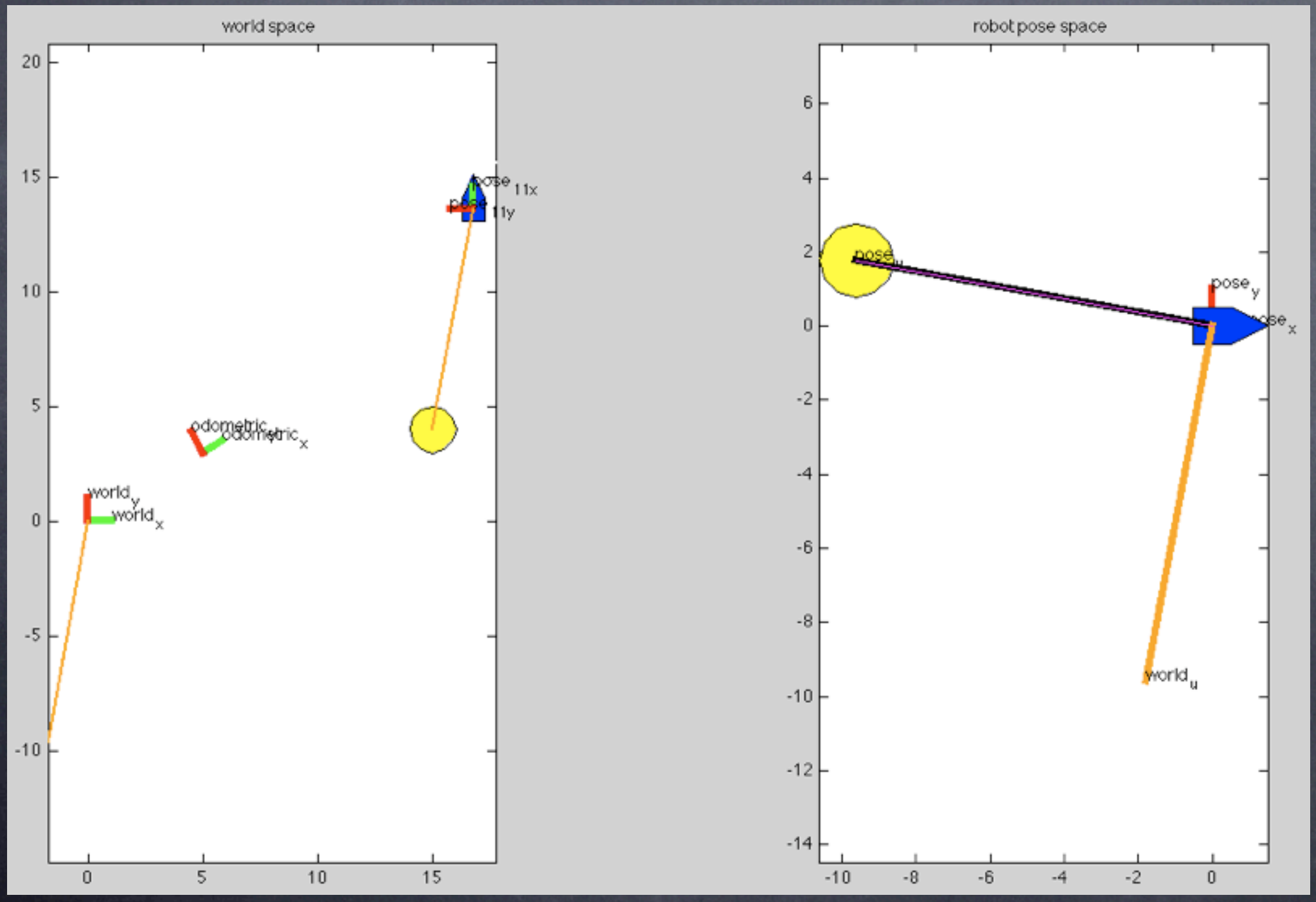
u^r

70

260

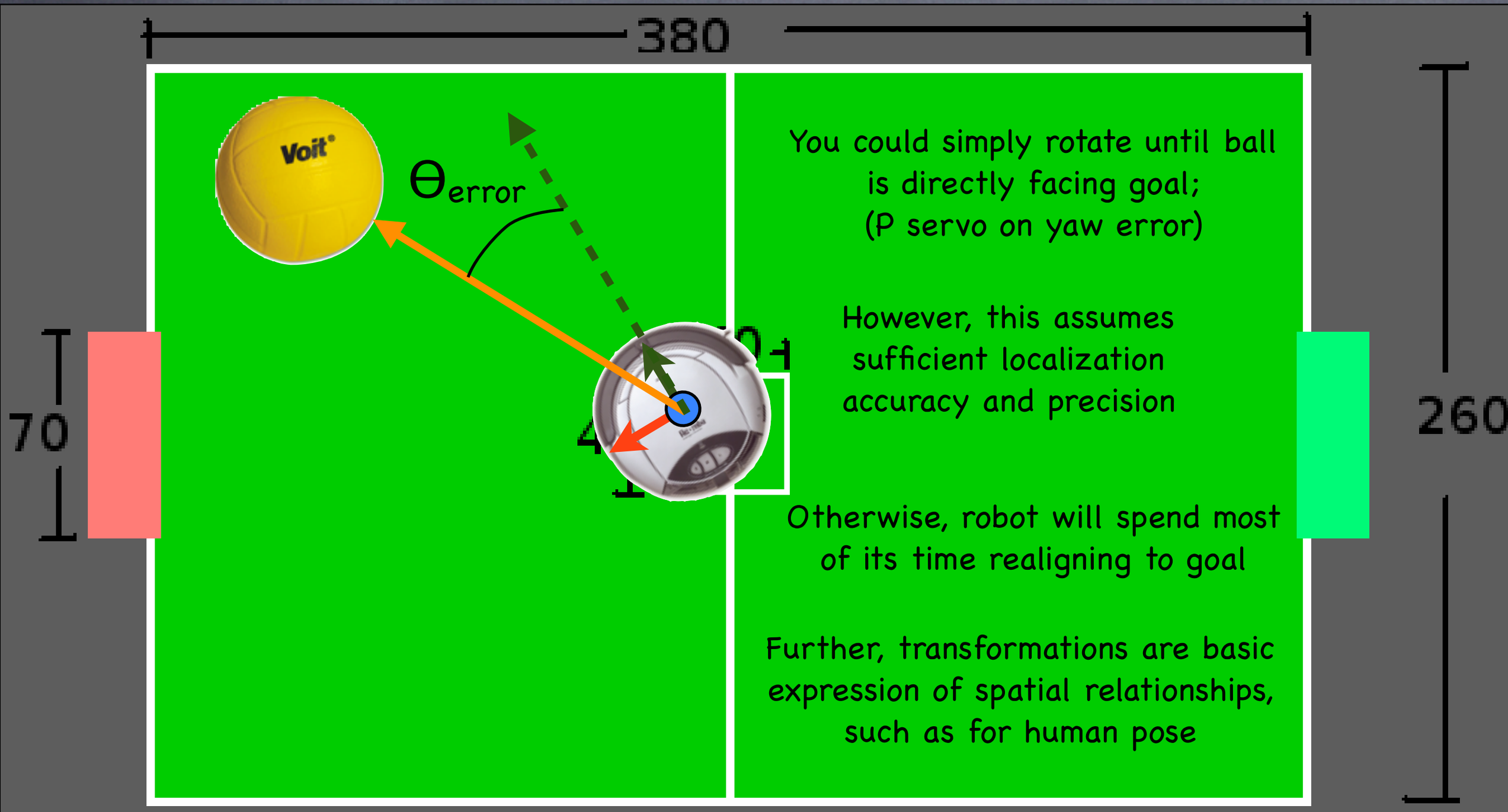
matlab example

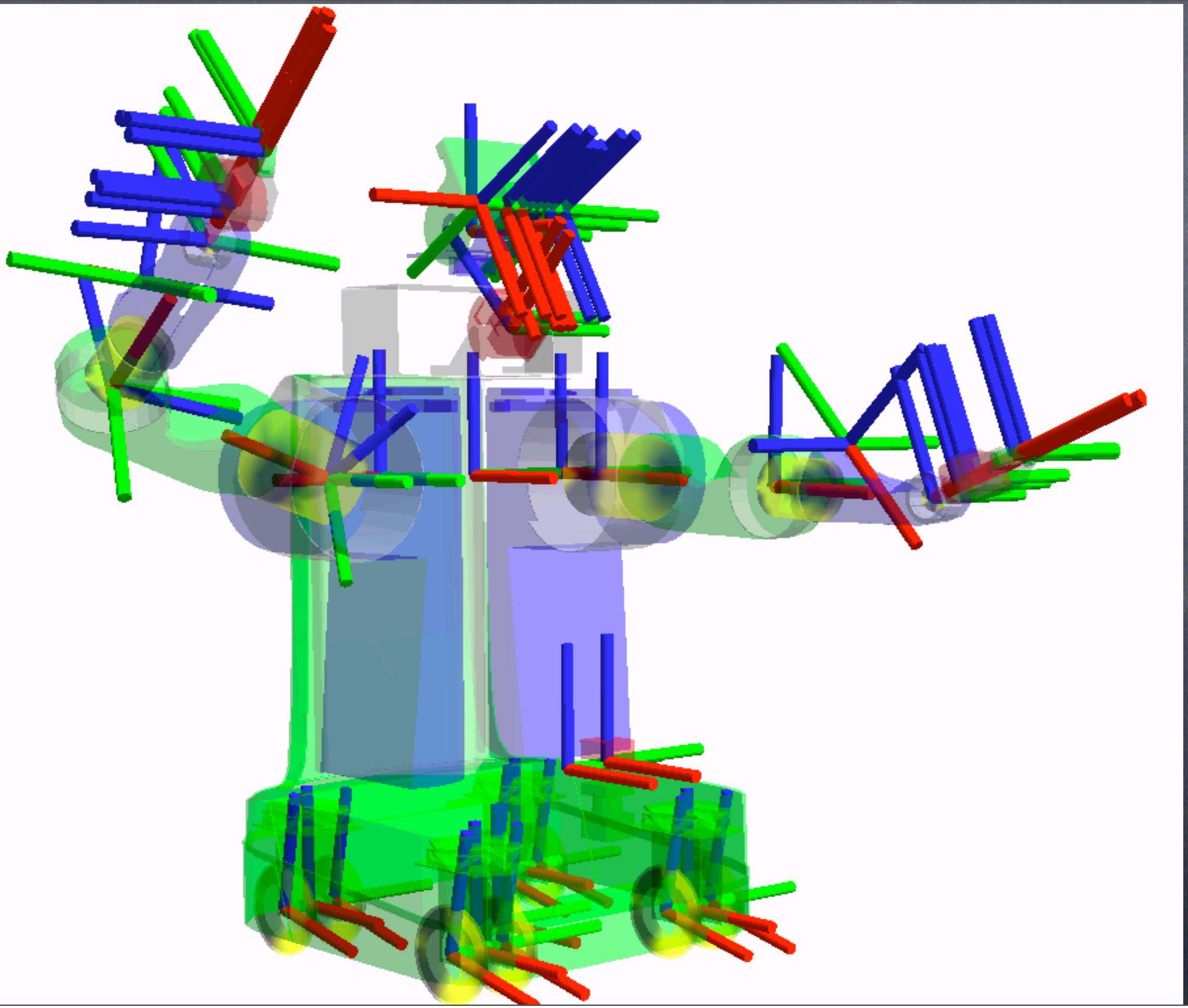
- /course/cs148/pub/odometry_coordinate_transform.m



with odometry

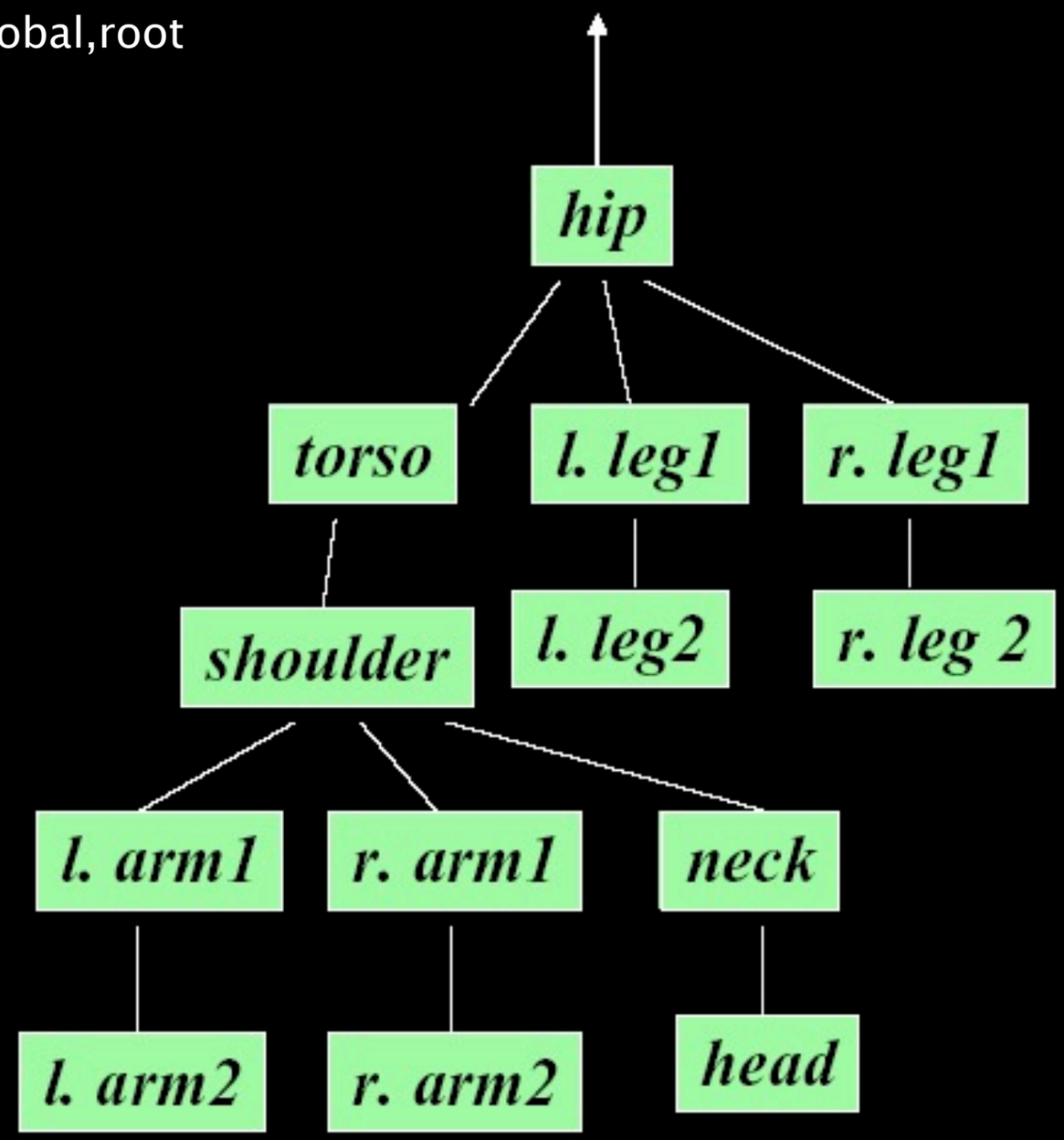
Why bother with transforms?

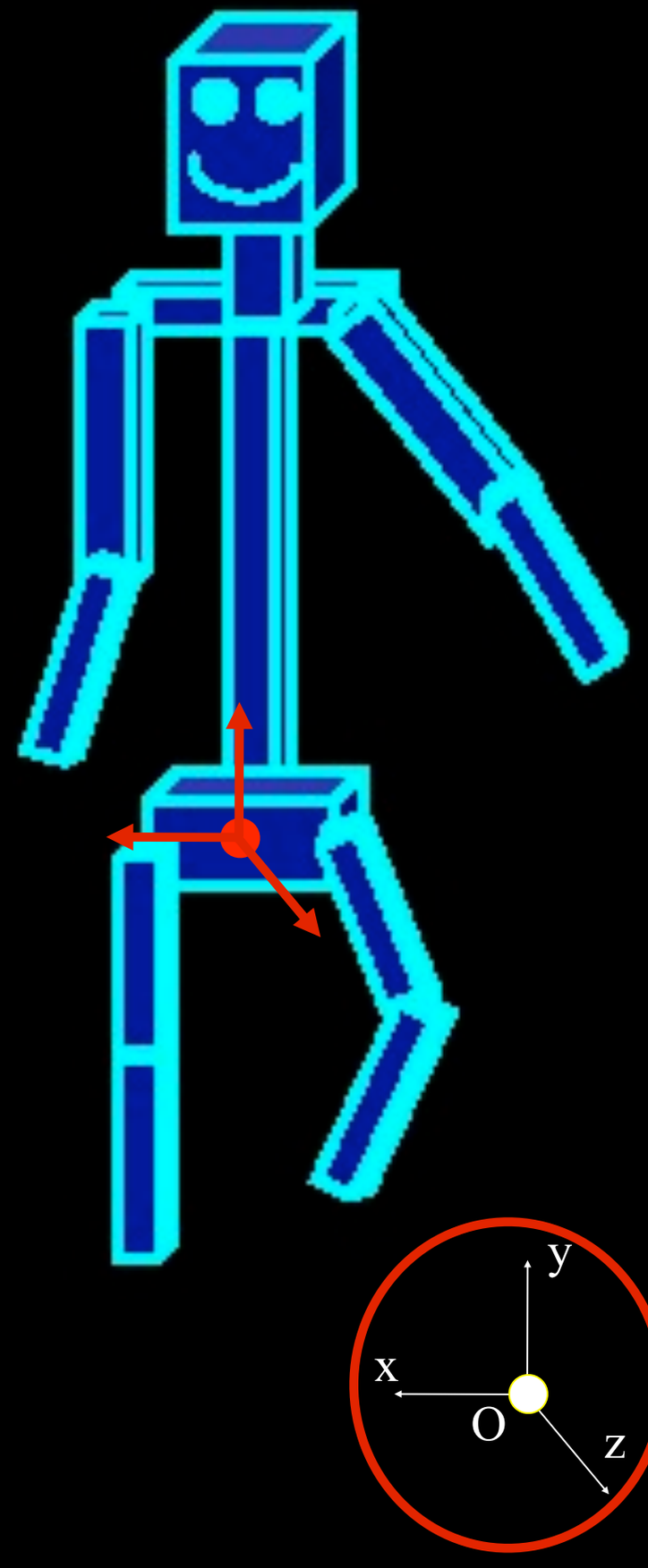




Human figure kinematics

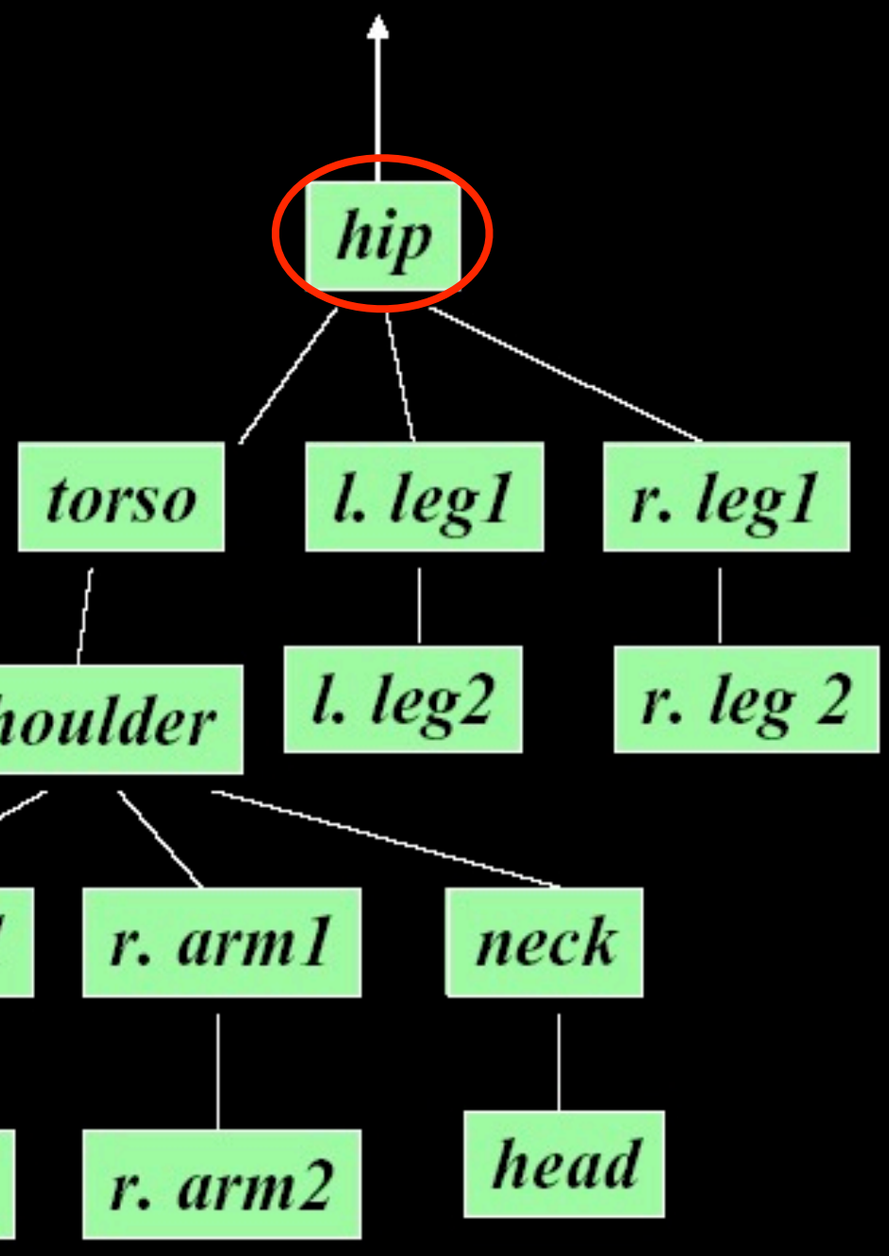
- Global coodinates $T_{global,root}$
  - absolute root

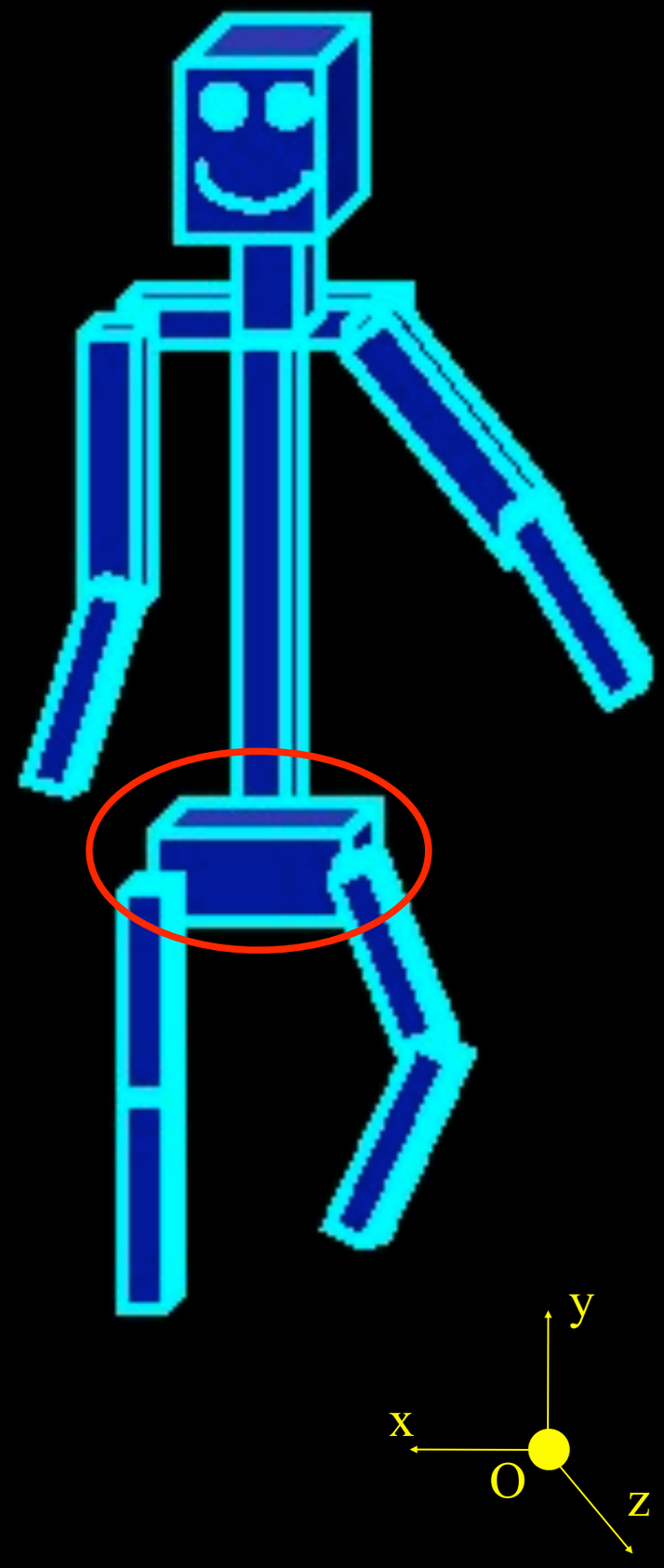




Human figure kinematics

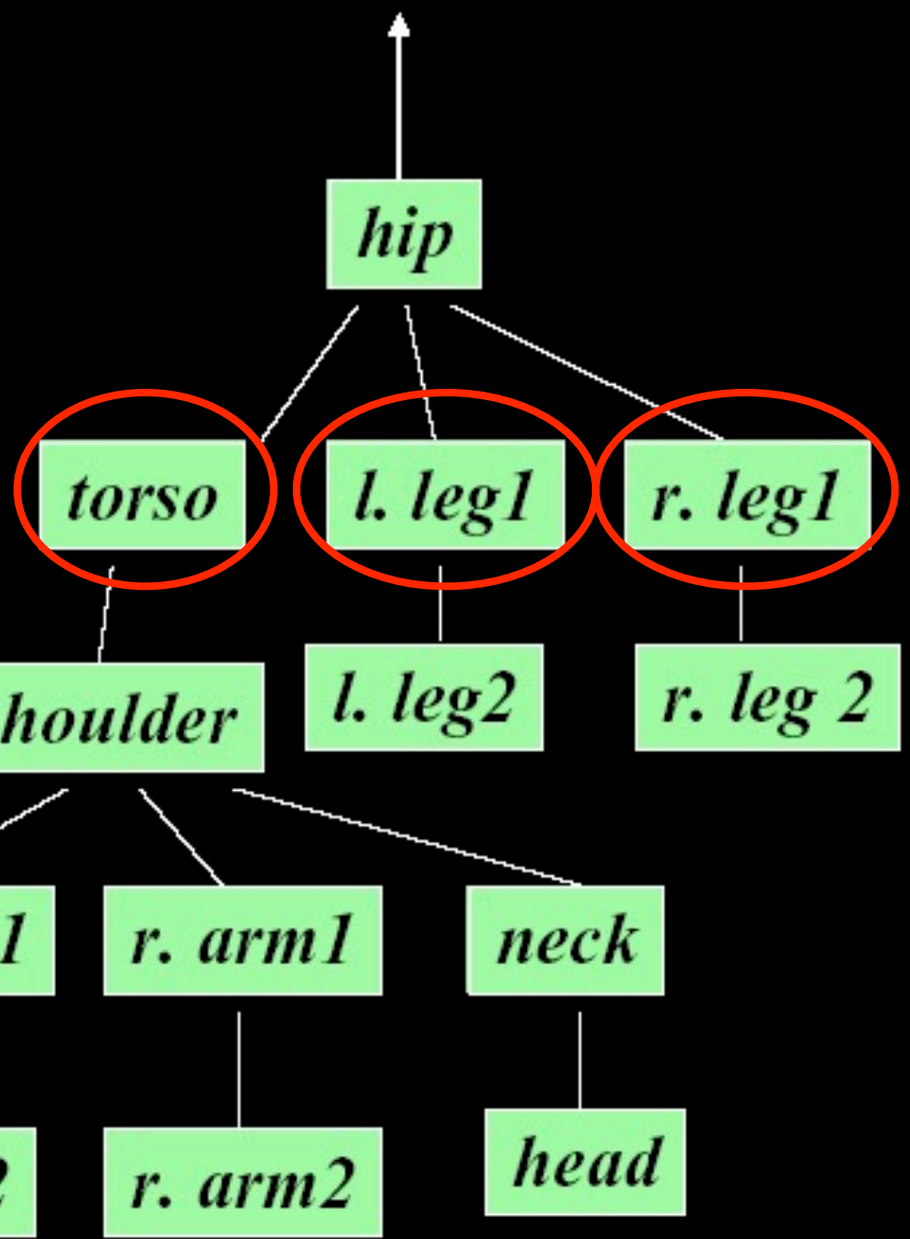
- Global coodinates $T_{global,root}$
  - absolute root
- Body root

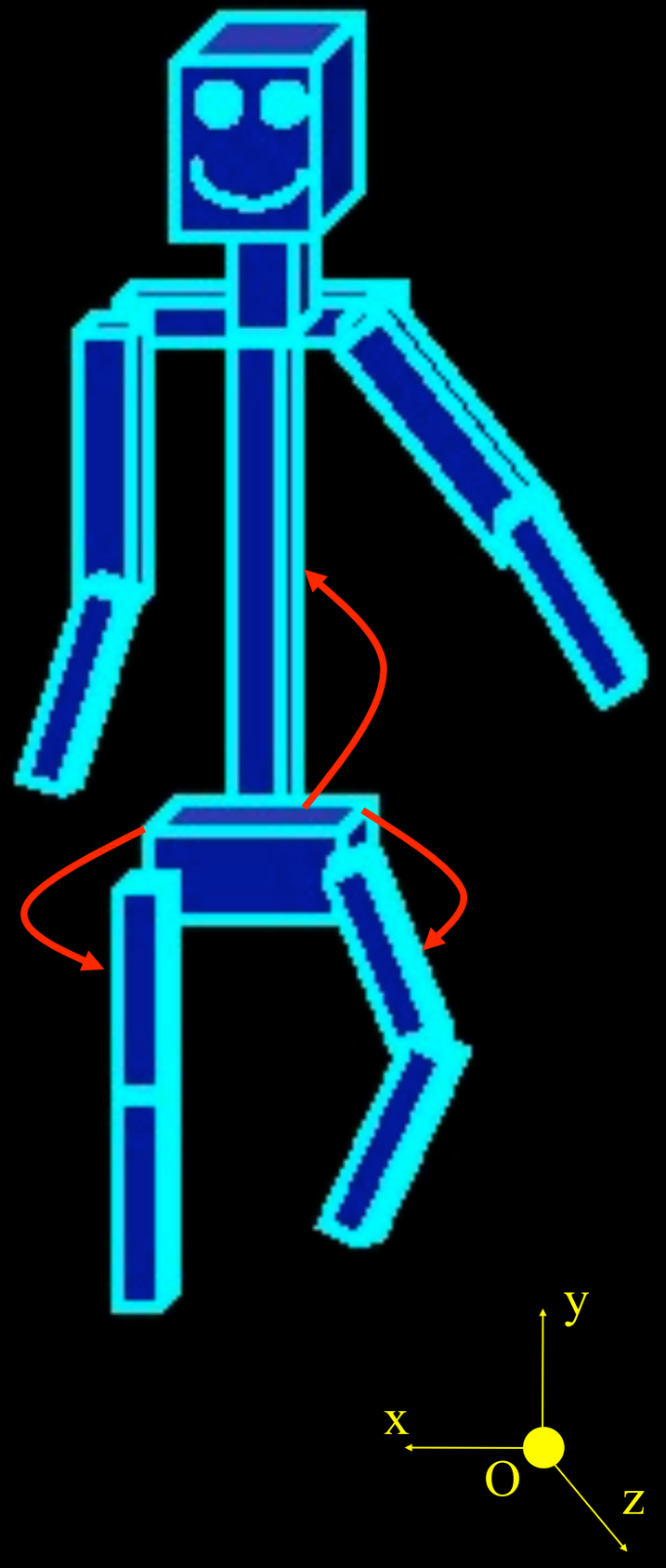




Human figure kinematics

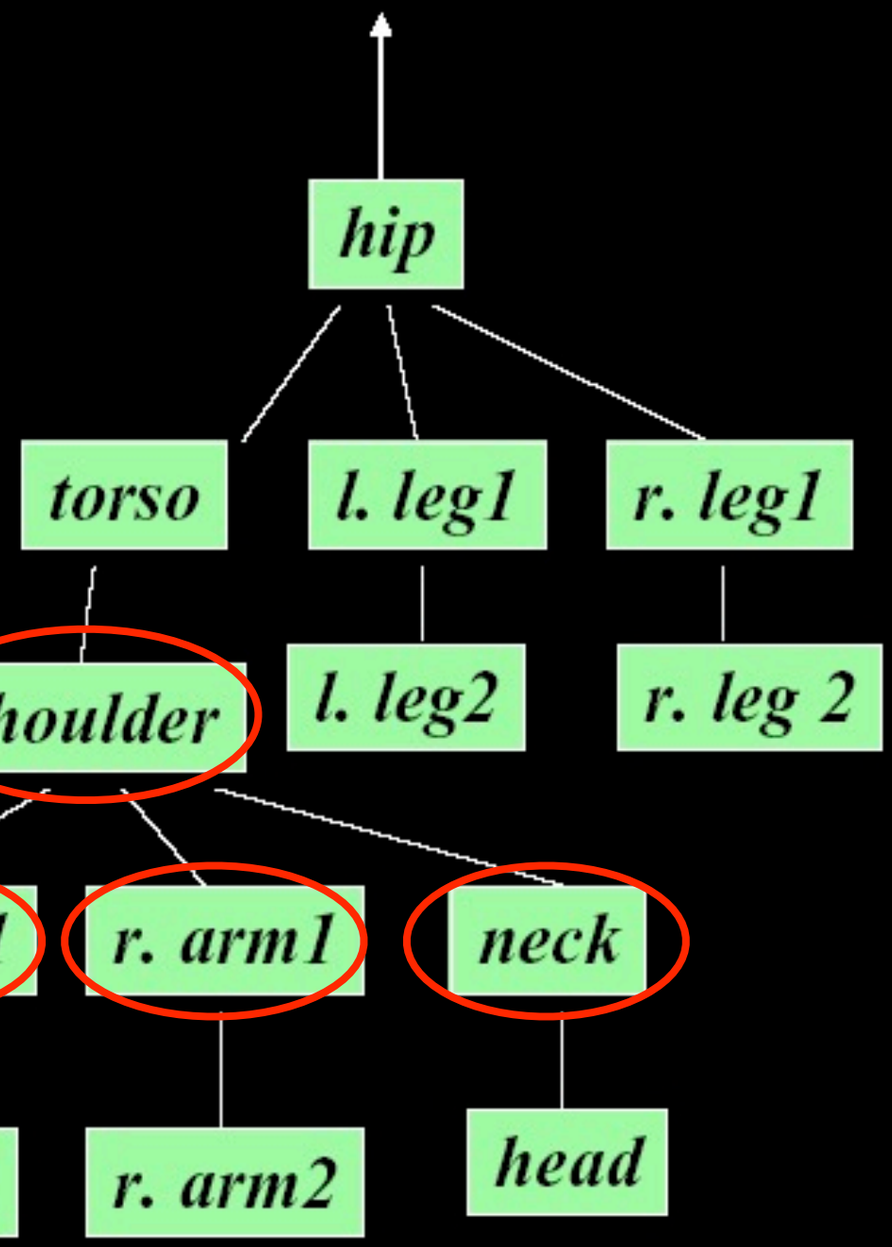
- Global coodinates $T_{global,root}$
  - absolute root
- Body root
- 1st level children

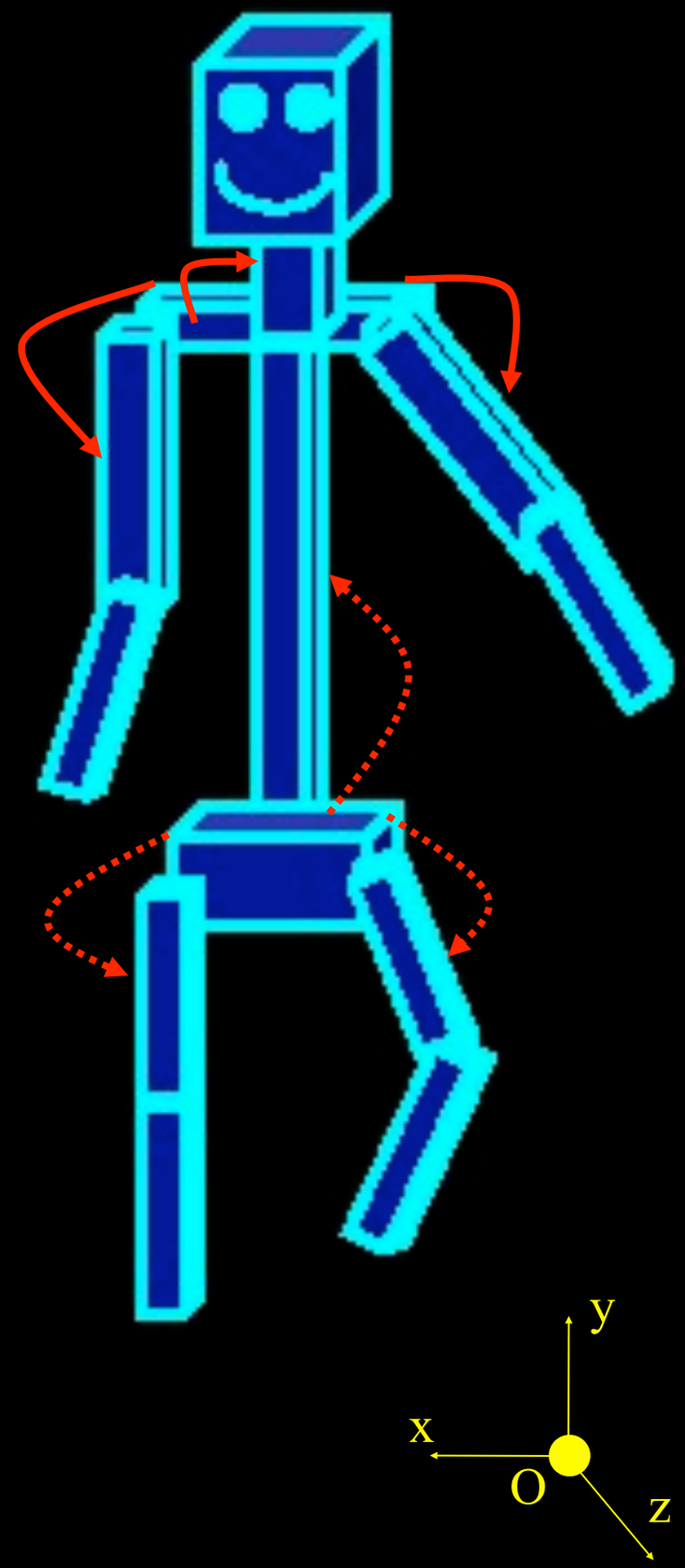




Human figure kinematics

- Global coodinates $T_{global,root}$
  - absolute root
- Body root
- 1st level children $T_{root,child}$
- Nth level children $T_{parent,child}$

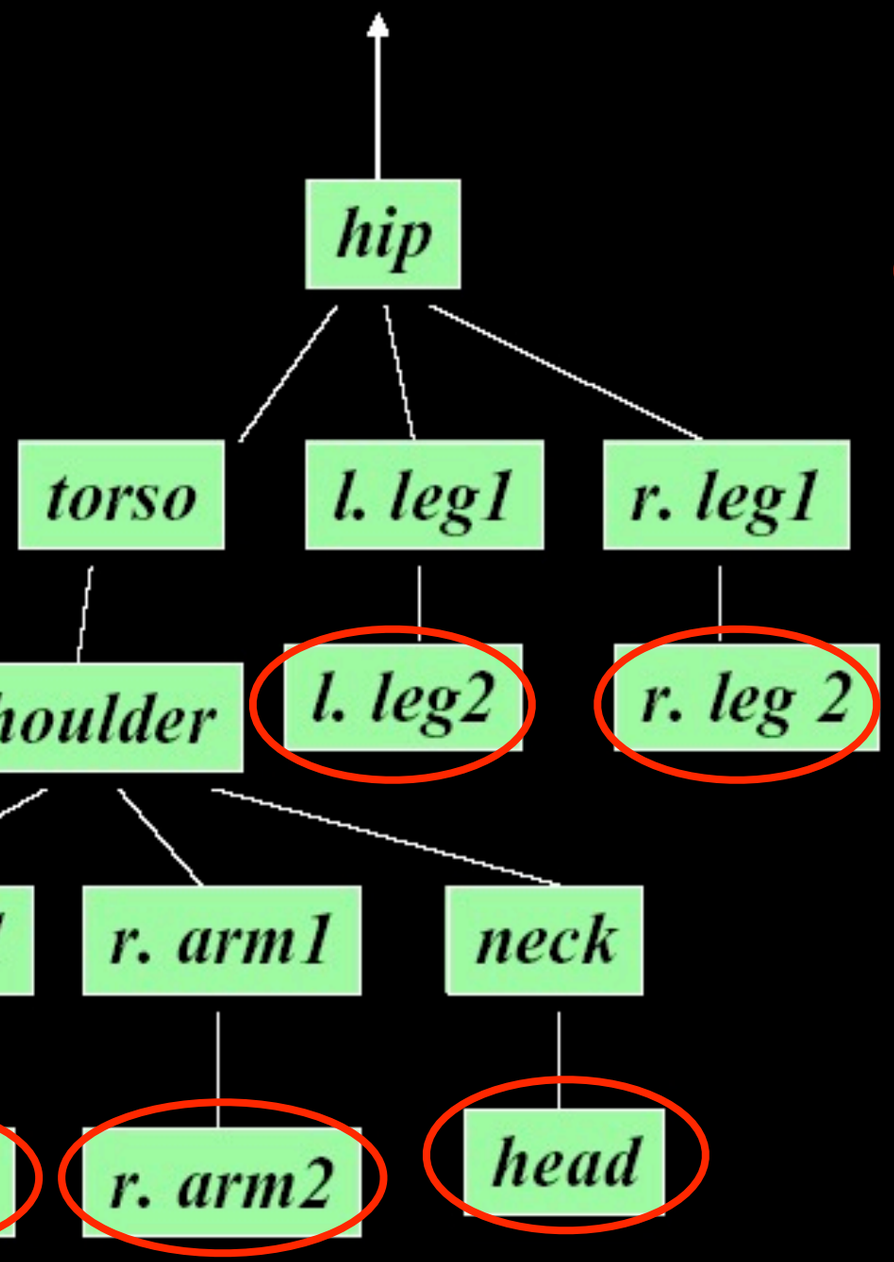


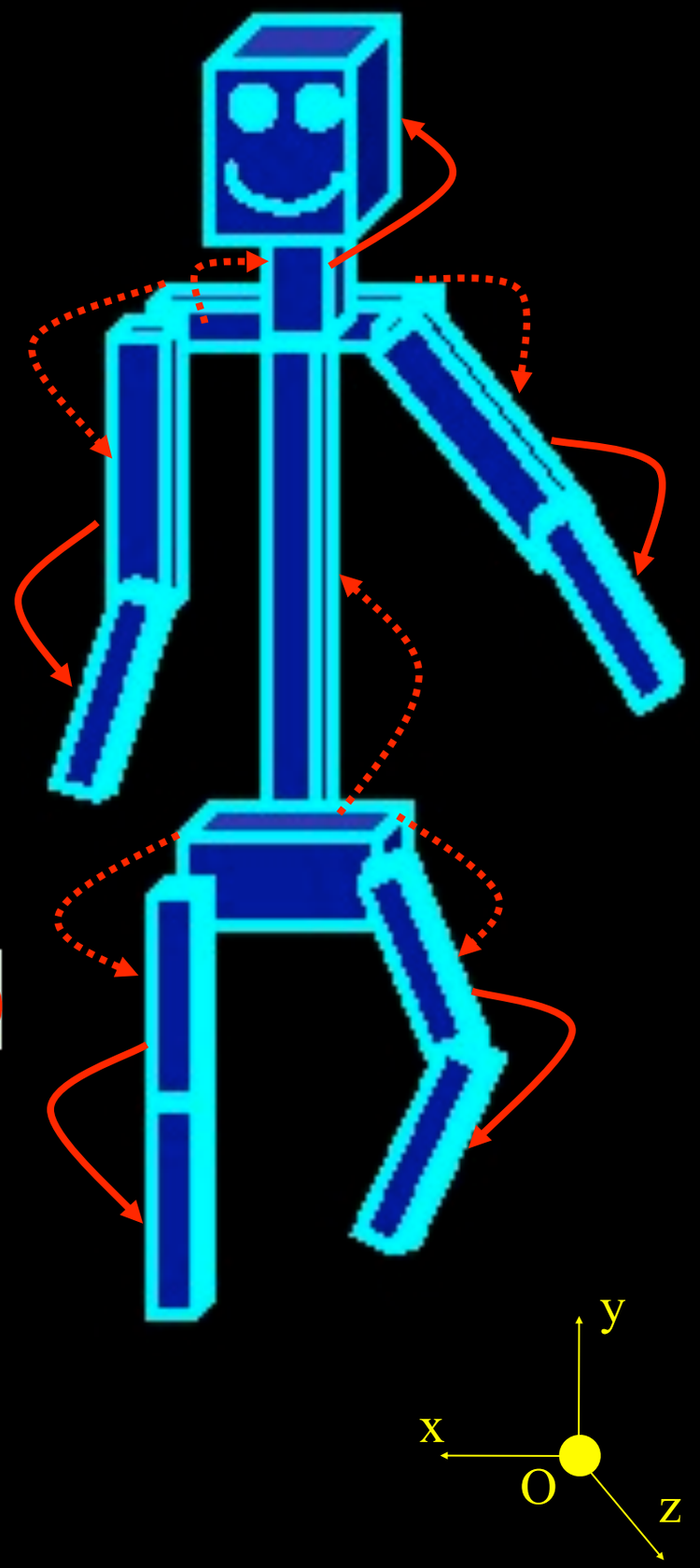


Human figure kinematics

- Global coodinates $T_{global,root}$
  - absolute root
- Body root
- 1st level children $T_{root,child}$
- Nth level children $T_{parent,child}$
- Leaf bodies

  $T_{parent,leaf}$





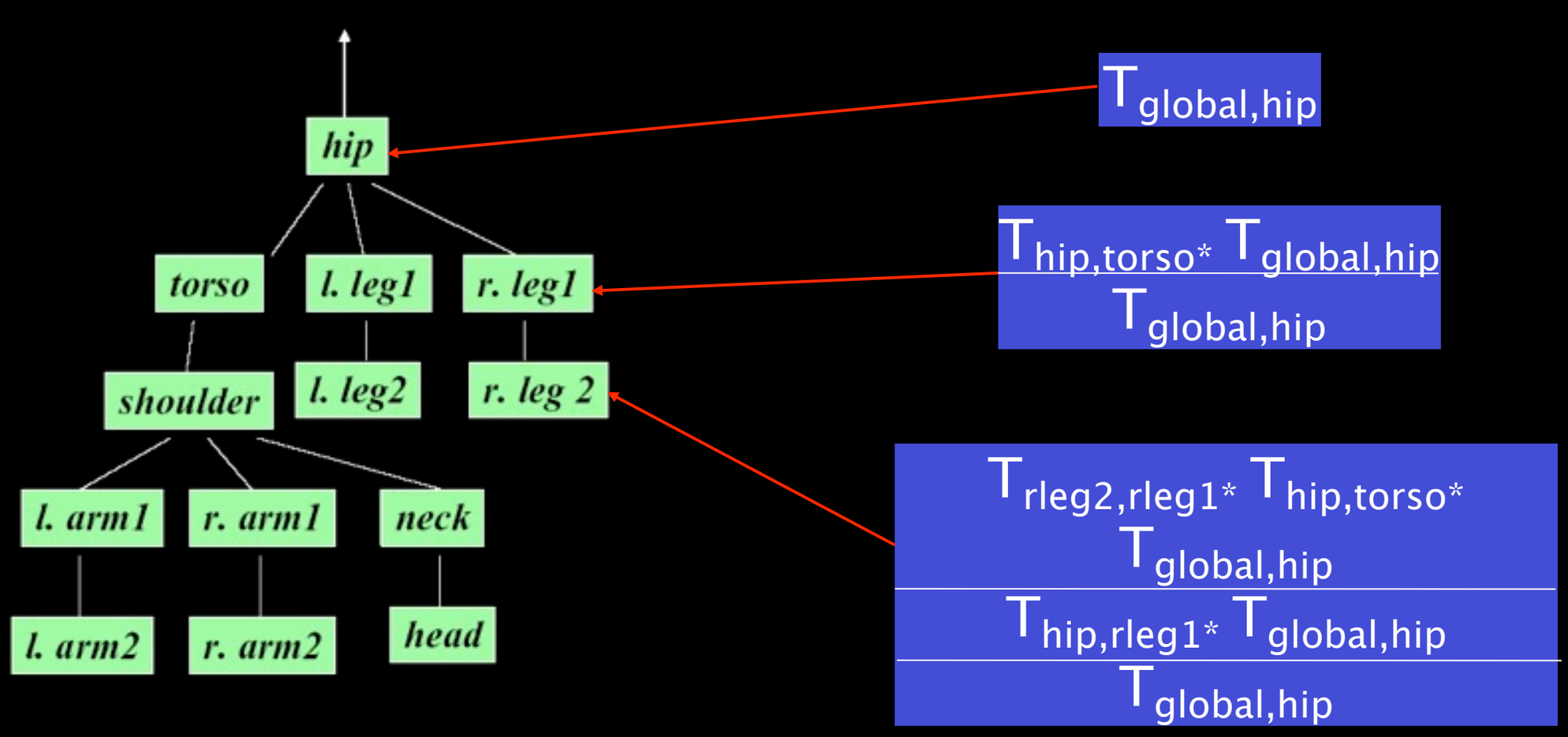
Hierarchical kinematic representations

- Hierarchical kinematic representations define the configuration space of the robot
  - a robot configuration is specified by a vector containing all of the robot's degrees-of-freedom (DOFs)
  - think of a D-dimensional space where each DOF is an axis

- Recursive notations for hierarchical kinematics
  - Matrix stack
  - Denavit-Hartenberg notation

Kinematic matrix stack

- Maintain global transformation into current local coordinates at the top of a stack
- Push transformation onto stack when entering a local frame
- Pop transformation from stack when leaving a local frame



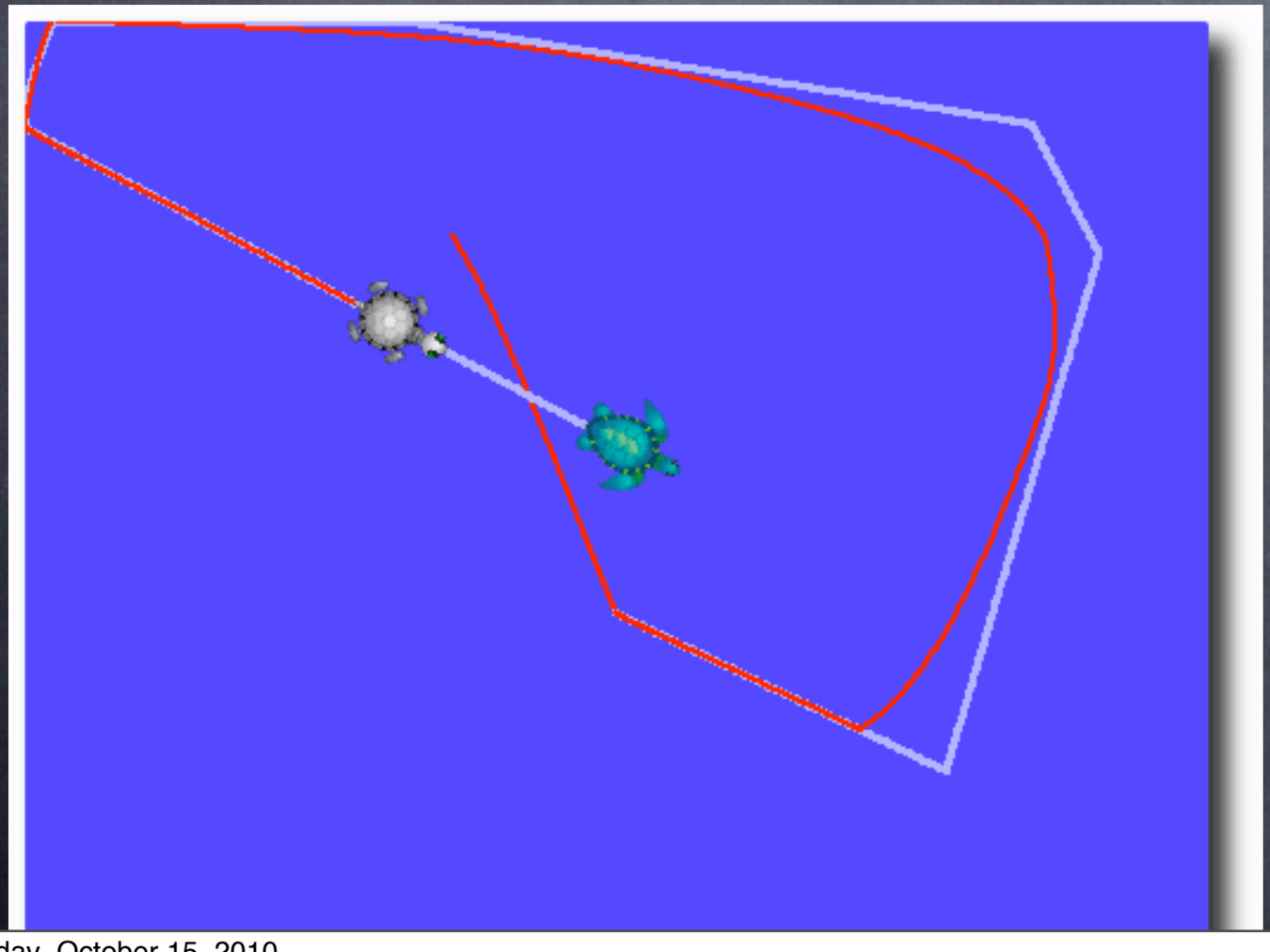
ROS tf Tutorials

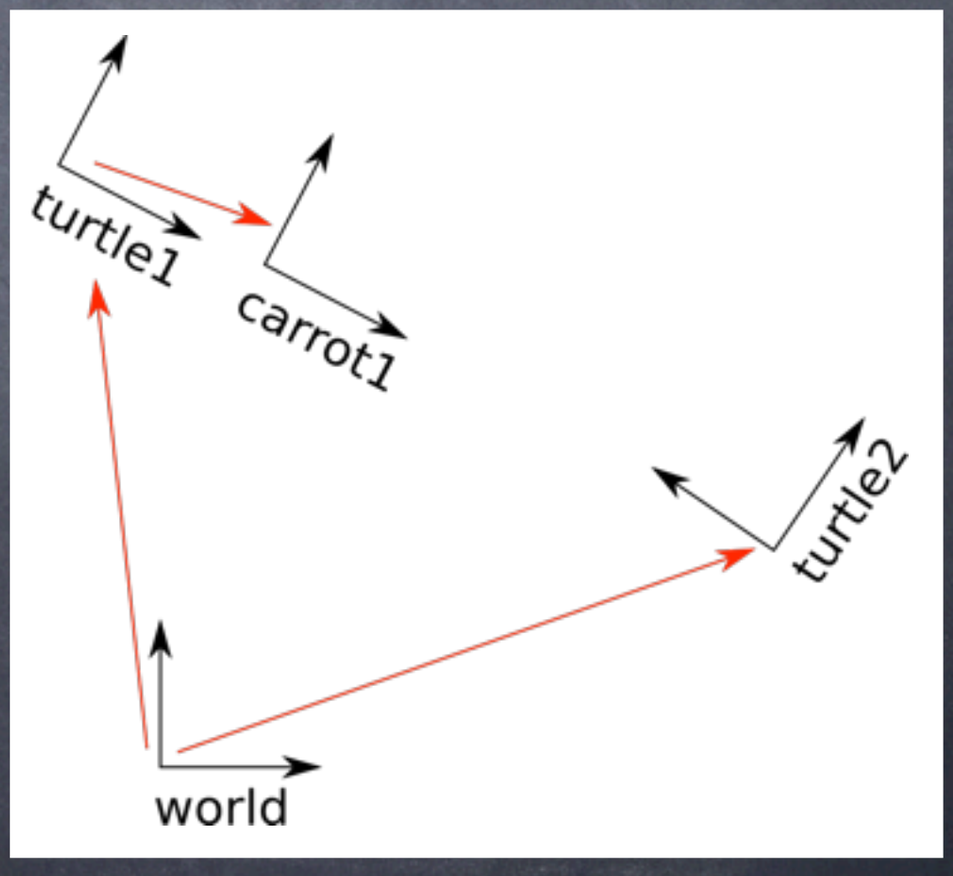
Do not use tf for projects!

tf: package for maintaining coordinate frames over time

Introduction to tf tutorial:
http://www.ros.org/wiki/tf/Tutorials/Introduction%20to%20tf

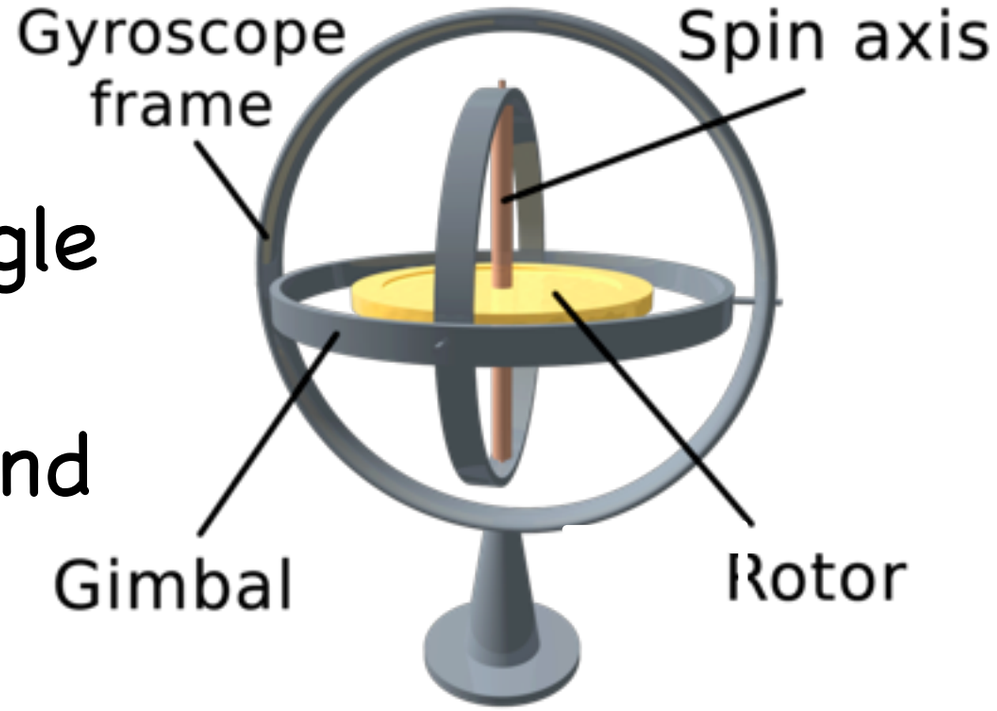
Publishing odometry using tf
http://www.ros.org/wiki/navigation/Tutorials/RobotSetup/Odom

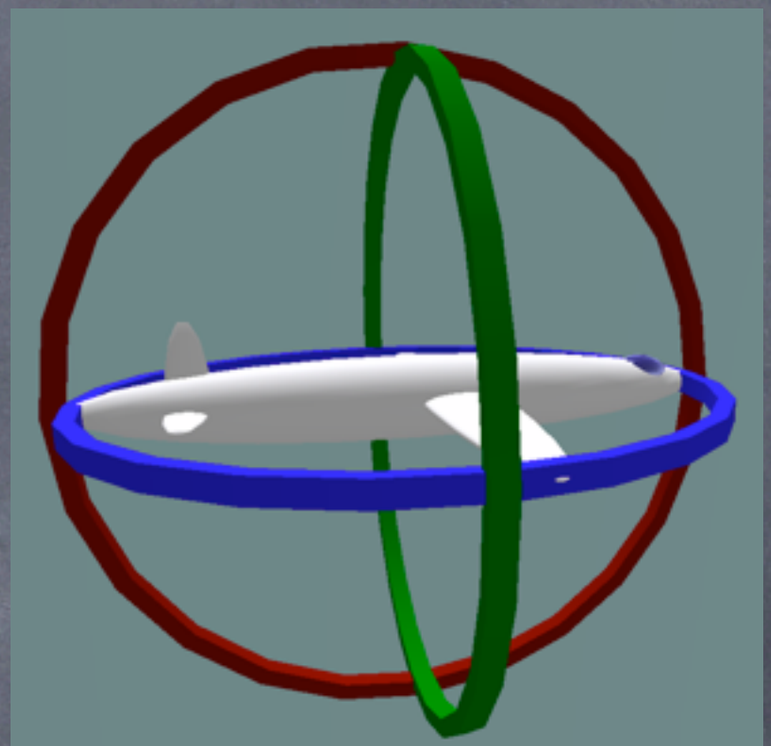




Gimbal Lock

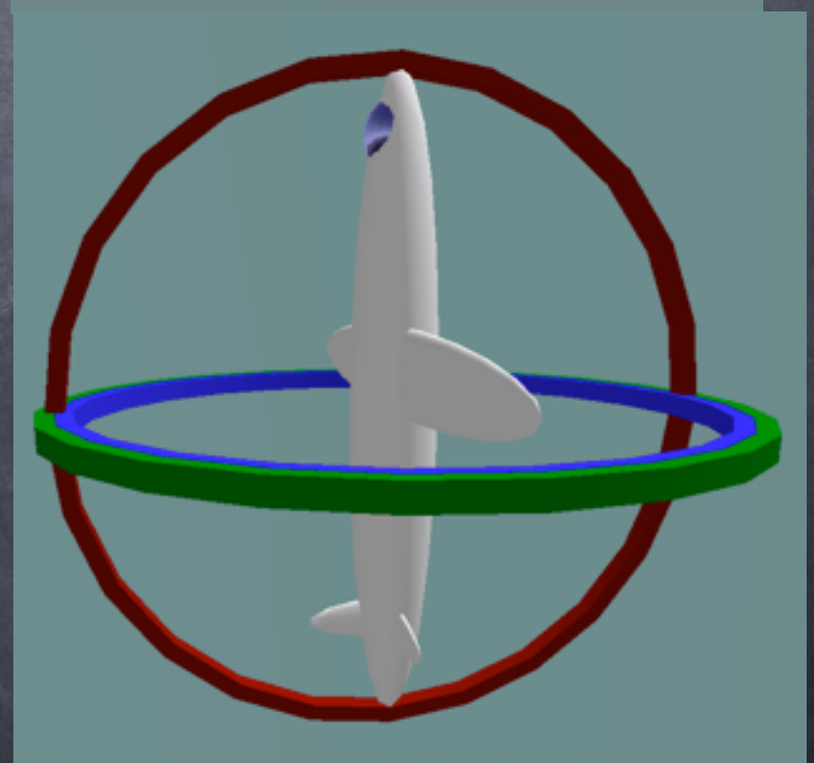
Gyroscope measures angle on each axis (wrt frame and wheel)





Gimbal lock occurs when two axes are rotated into alignment

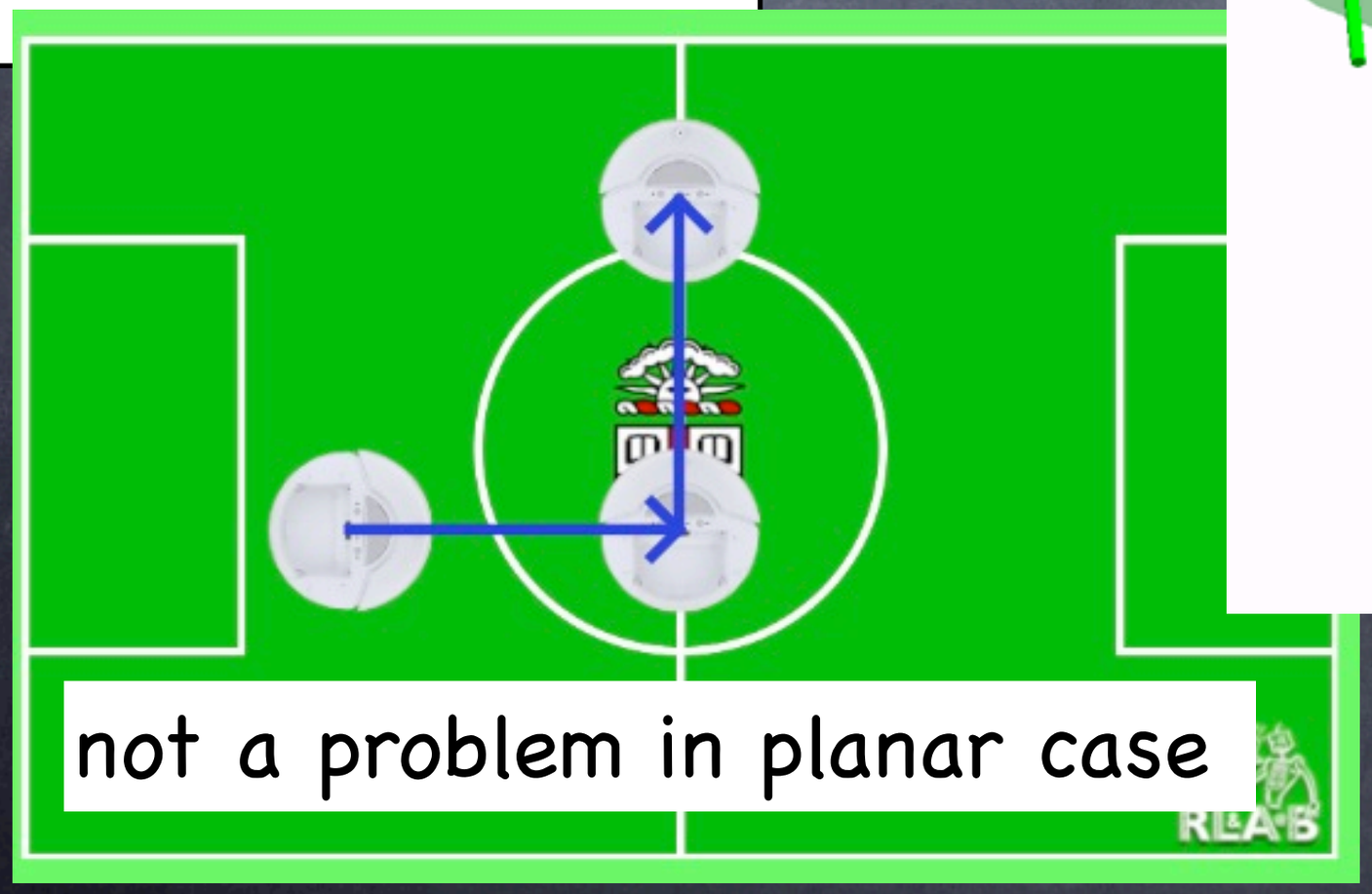
Reduces 3 DOFs to 2.
Based on axis order.



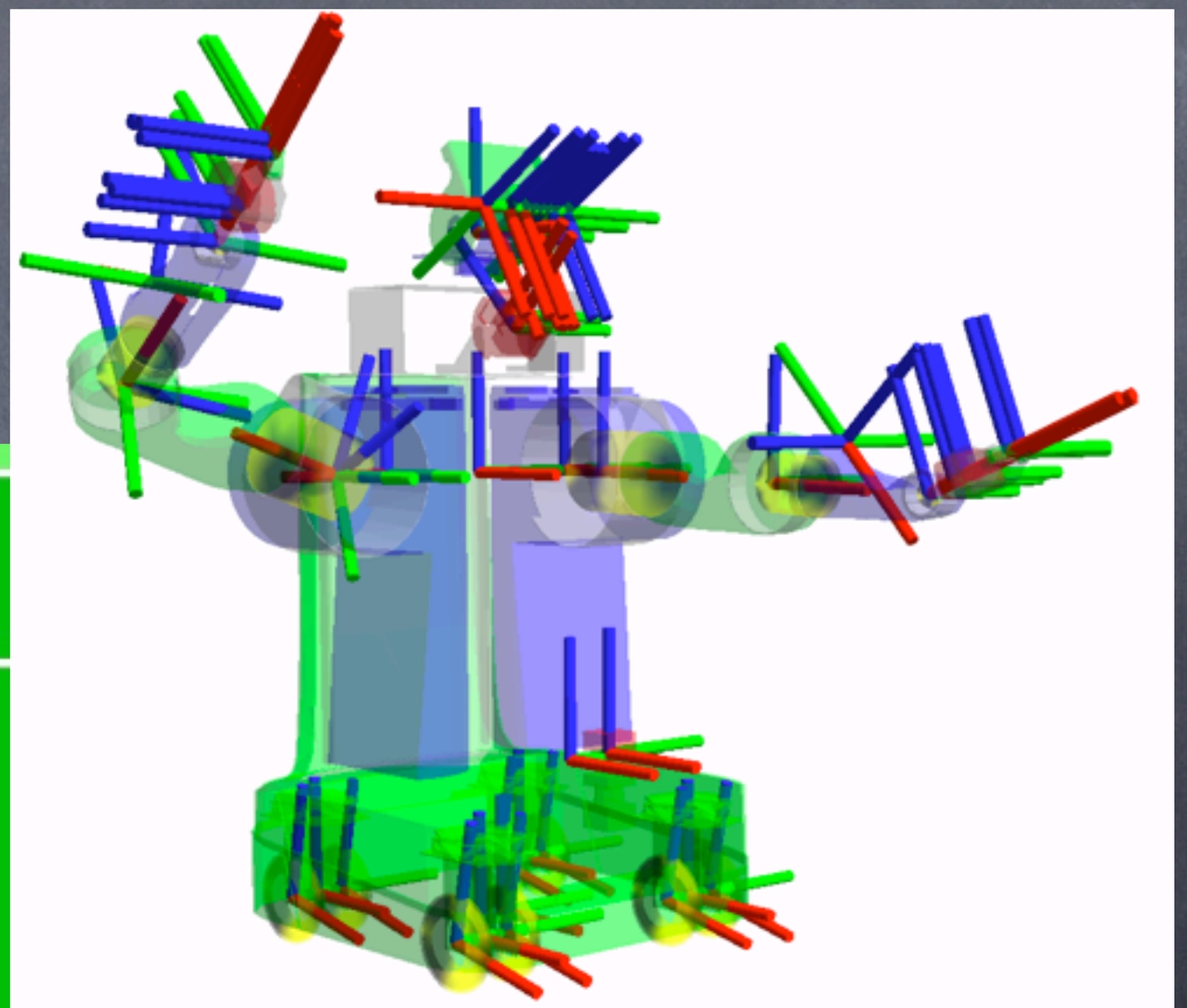
Gimbal Lock and JointTrajectory

JointTrajectory message:

```
Header header
    uint32 seq
    time stamp
    string frame_id
string[] joint_names
JointTrajectoryPoint[] points
    float64[] positions
    float64[] velocities
    float64[] accelerations
    duration time_from_start
```



not a problem in planar case



problem for 3D rotation

JointTrajectory example for planning project

File Edit View History Bookmarks Tools Help

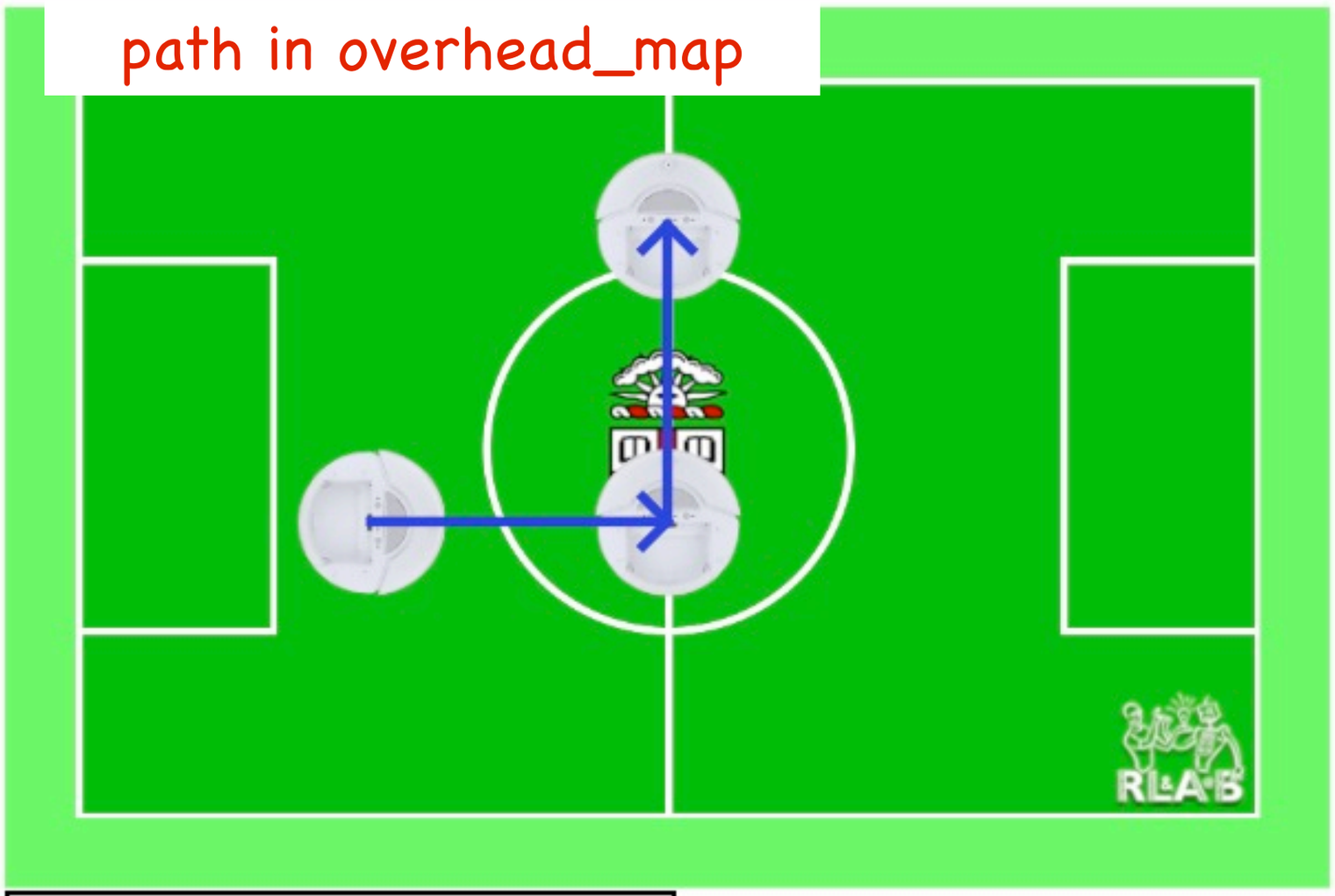
file:///opt/ros/cturtle/ros/brown-ros-pkg/overhead_map/overhead_m

2d transforms

Feedback

7. Compound ... | Global variabl... | Google Docs -... | Problem loadi... | brown-ros-pk.

path in overhead_map



JointTrajectory definition

JointTrajectory message:

Header header
  uint32 seq
  time stamp
  string frame_id
string[] joint_names
JointTrajectoryPoint[] points
  float64[] positions
  float64[] velocities
  float64[] accelerations
  duration time_from_start

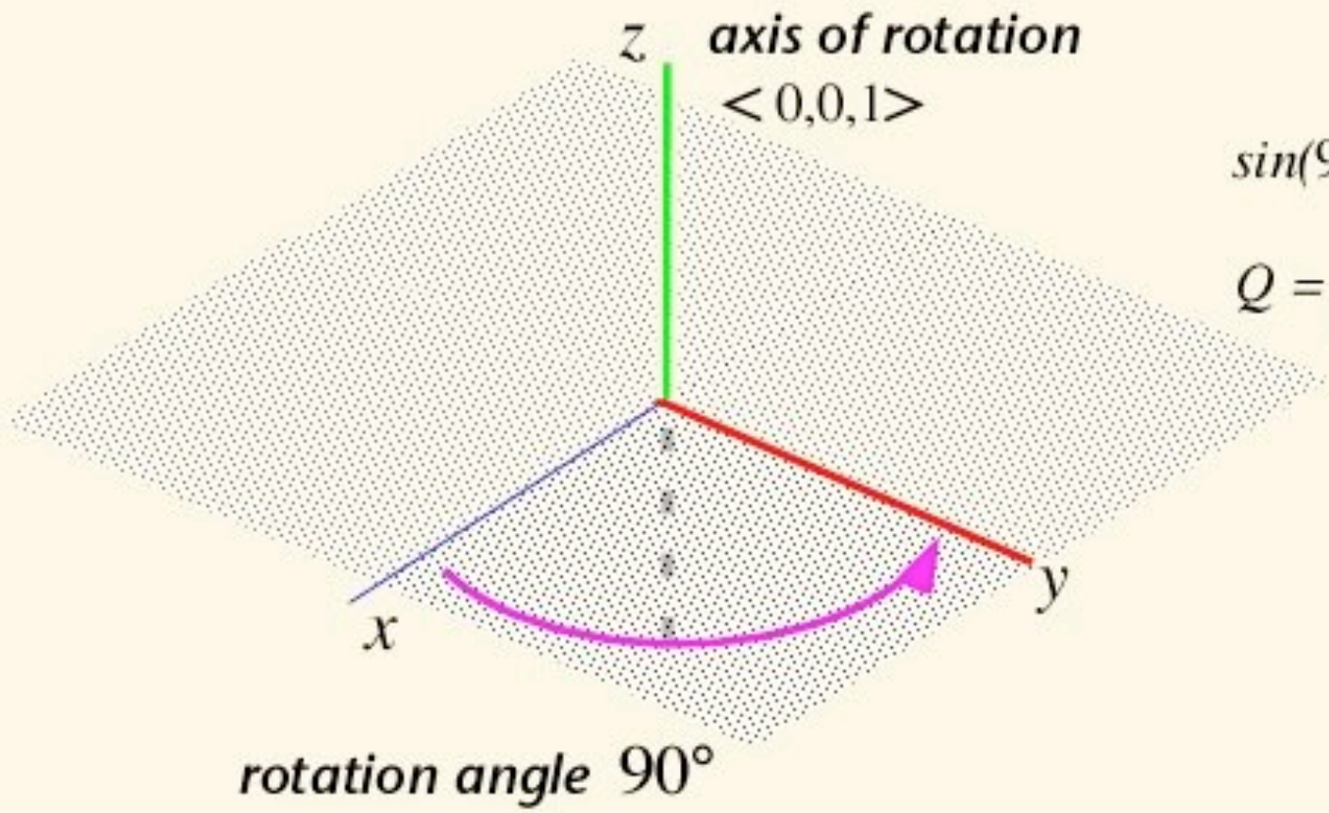
Done

Rotation by Quaternion

(No axis order, No gimbal lock)

3D rotation can be expressed by an axis and an angle

Quaternions provide a clean mathematical expression



$$\sin(90°/2)=\cos(90°/2)=1/\sqrt{2}$$

$$Q = \frac{1}{\sqrt{2}}\langle 0,0,1,1\rangle$$

vector part

scalar part

Quaternions

- Expressed as: $a + bi + cj + dk$
  - i, j, k satisfy $i^2 = j^2 = k^2 = ijk = -1$
  - a is scalar, (b,c,d) acts as rotation axis **u**

$$\mathbf{i} \equiv \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\mathbf{j} \equiv \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{k} \equiv \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{1} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Quaternion $q = a + bi + cj + dk$
  - $= a + (b,c,d) = \cos(\text{angle}/2) + u \sin(\text{angle}/2)$

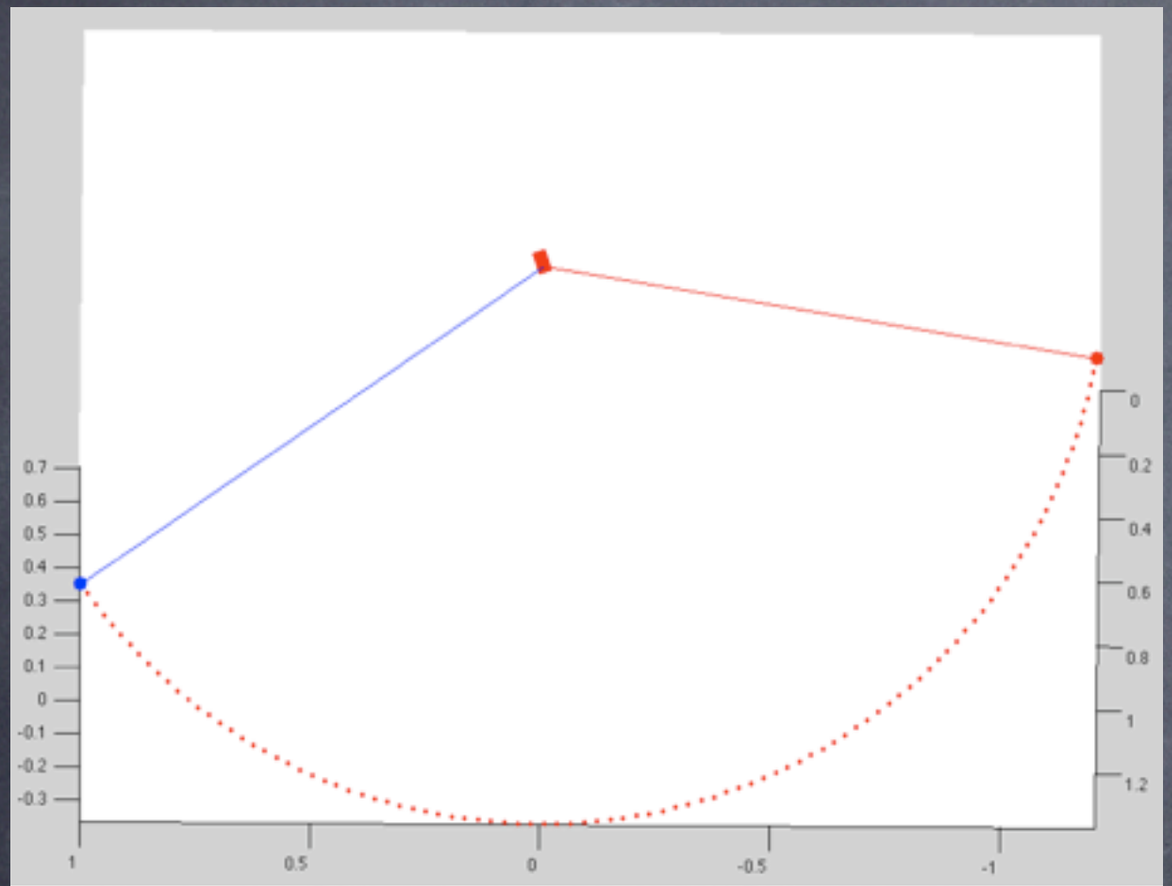
Quaternion Conjugation

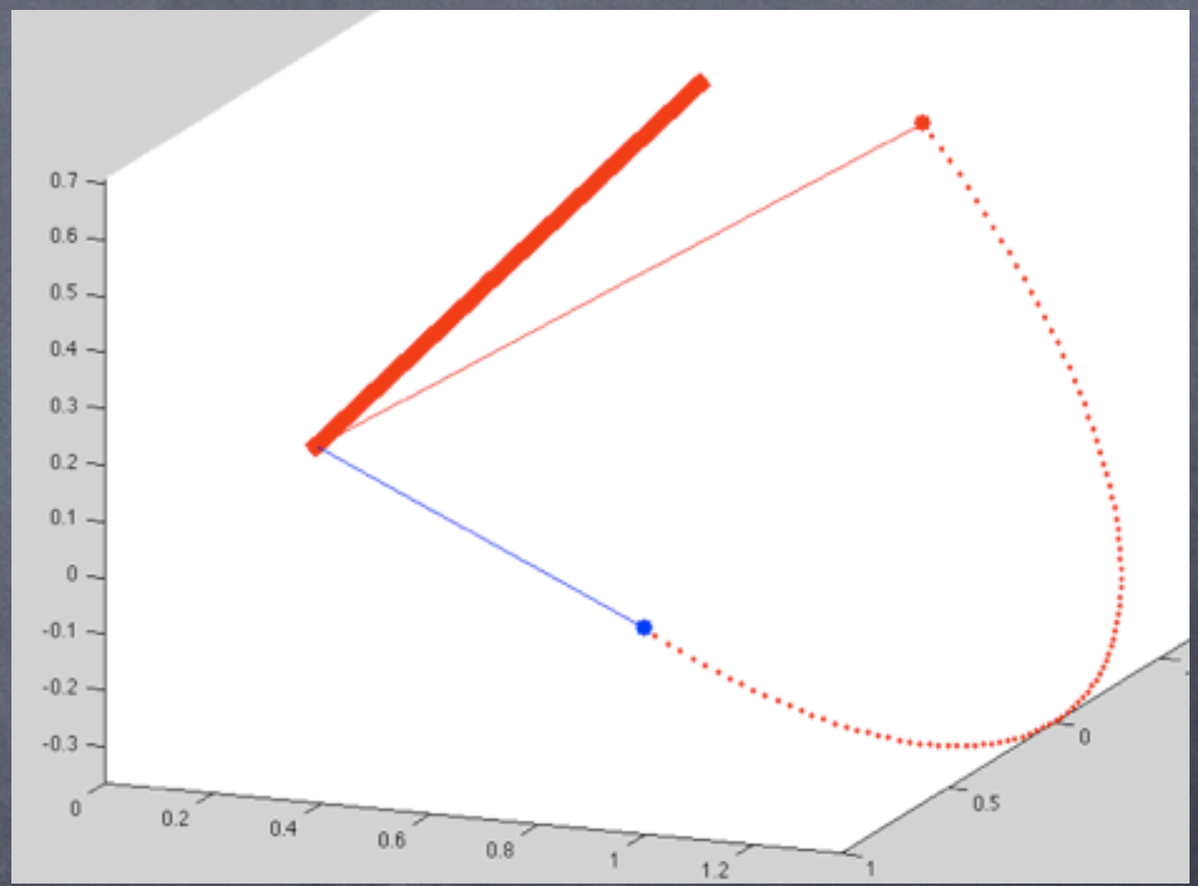
- Rotating a point **v** by unit quaternion q is performed by conjugation
  - $v' = qvq^{-1}$, where $q^{-1} = a - \mathbf{u}$, $v = 0 + \mathbf{v}$
    - q must be unit quaternion: ||q|| = 1
  - Multiplication of 2 quaternions
    - (a+bi+cj+dk)(e+fi+gk+hk) =

(ae-bf-cg-ah)+(af+be+ch-dg)**i**+(ag-bh+ce+df)**j**+(ah+bg-cf+de)**k**

Matlab example

http://www.cs.brown.edu/courses/cs148/pub/quaternion_rotation.m



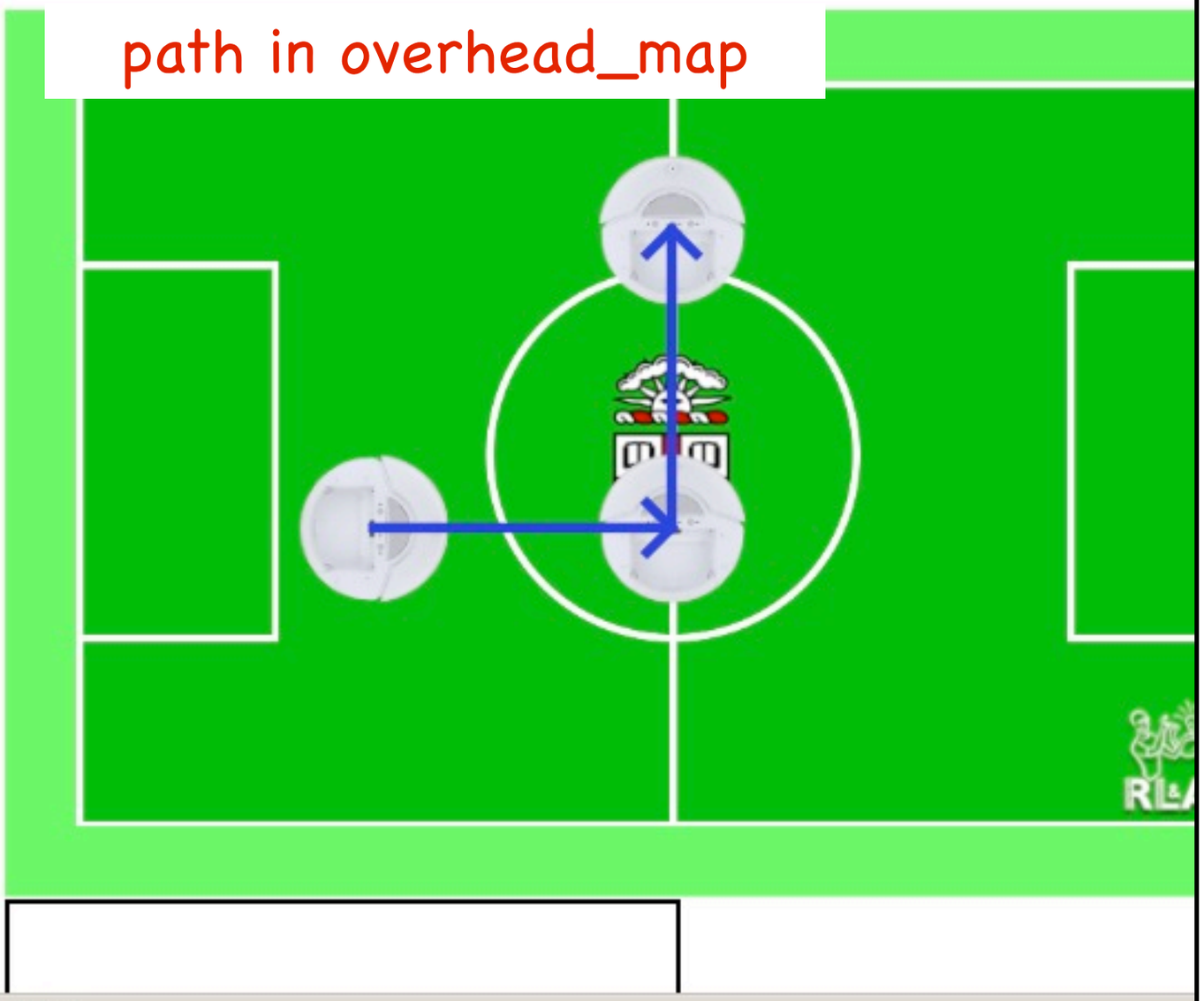


Rotation of point [1 1 0] by
axis [0 sin(pi/4) cos(pi/4)] and angle 3pi/4

File Edit View History Bookmarks Tools Help

file:///opt/ros/cturtle/ros/brown-ros-pkg/overhead_ma

7. Compound ... | Global variabl... | Google Docs -... | Problem loadi...



Done

Path message:

[Header](#) header
- uint32 seq
- time stamp
- string frame_id

[geometry_msgs/PoseStamped[]](#) poses
- [Header](#) header
  - uint32 seq
  - time stamp
  - string frame_id
- [geometry_msgs/Pose](#) pose
  - [geometry_msgs/Point](#) position
    - float64 x
    - float64 y
    - float64 z
  - [geometry_msgs/Quaternion](#) orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w