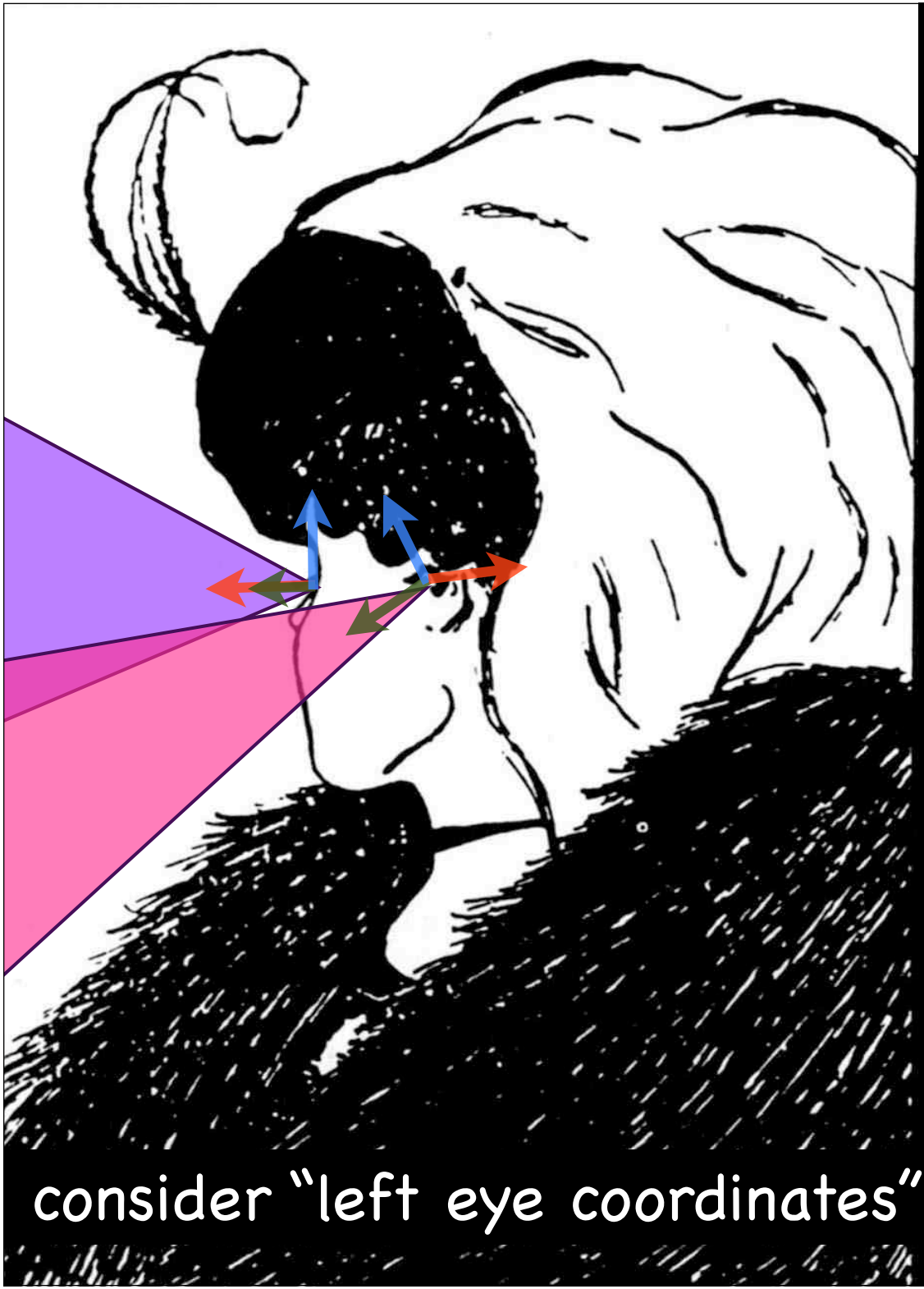
Topic 10

Coordinate Spaces and Transforms

It depends on your perspective



Topic 10

Coordinate Spaces and Transforms

It depends on your perspective

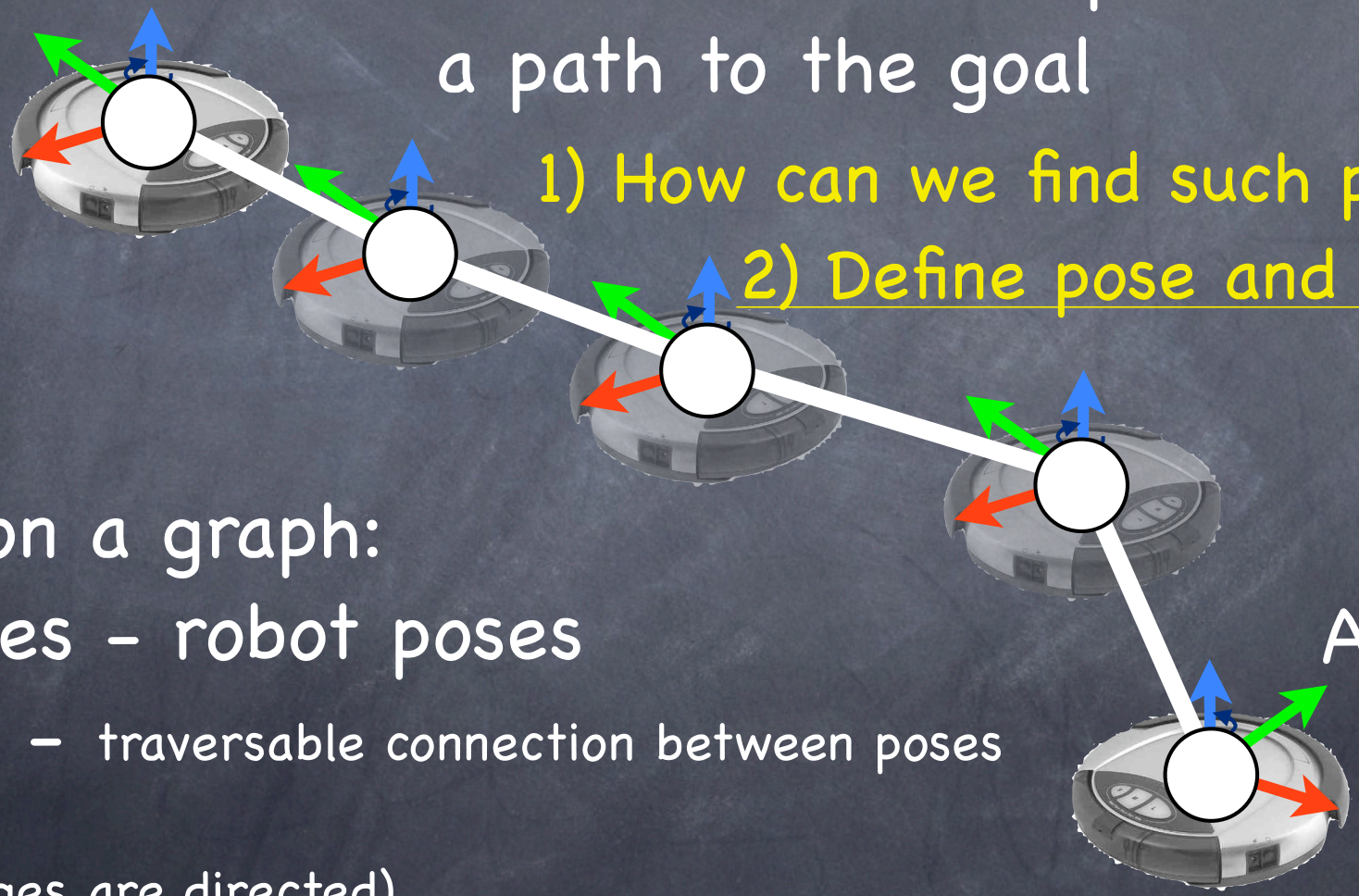
Path Planning

B: Goal

Find intermediate poses forming a path to the goal

1) How can we find such paths?

2) Define pose and controls?



Path on a graph:

vertices - robot poses

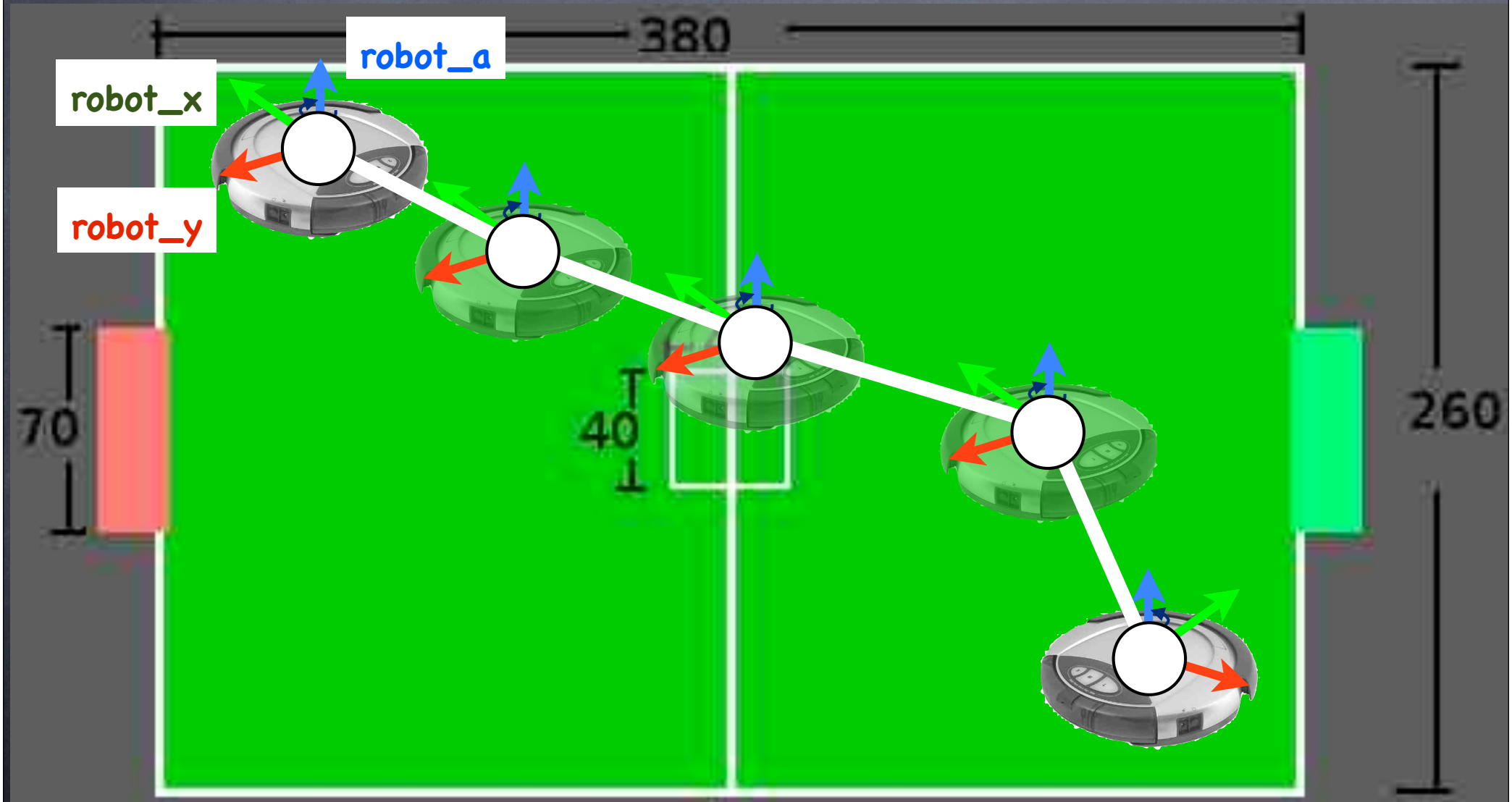
edges - traversable connection between poses

A: Start

(note edges are directed)

1) path planning: sequence of intermediate poses

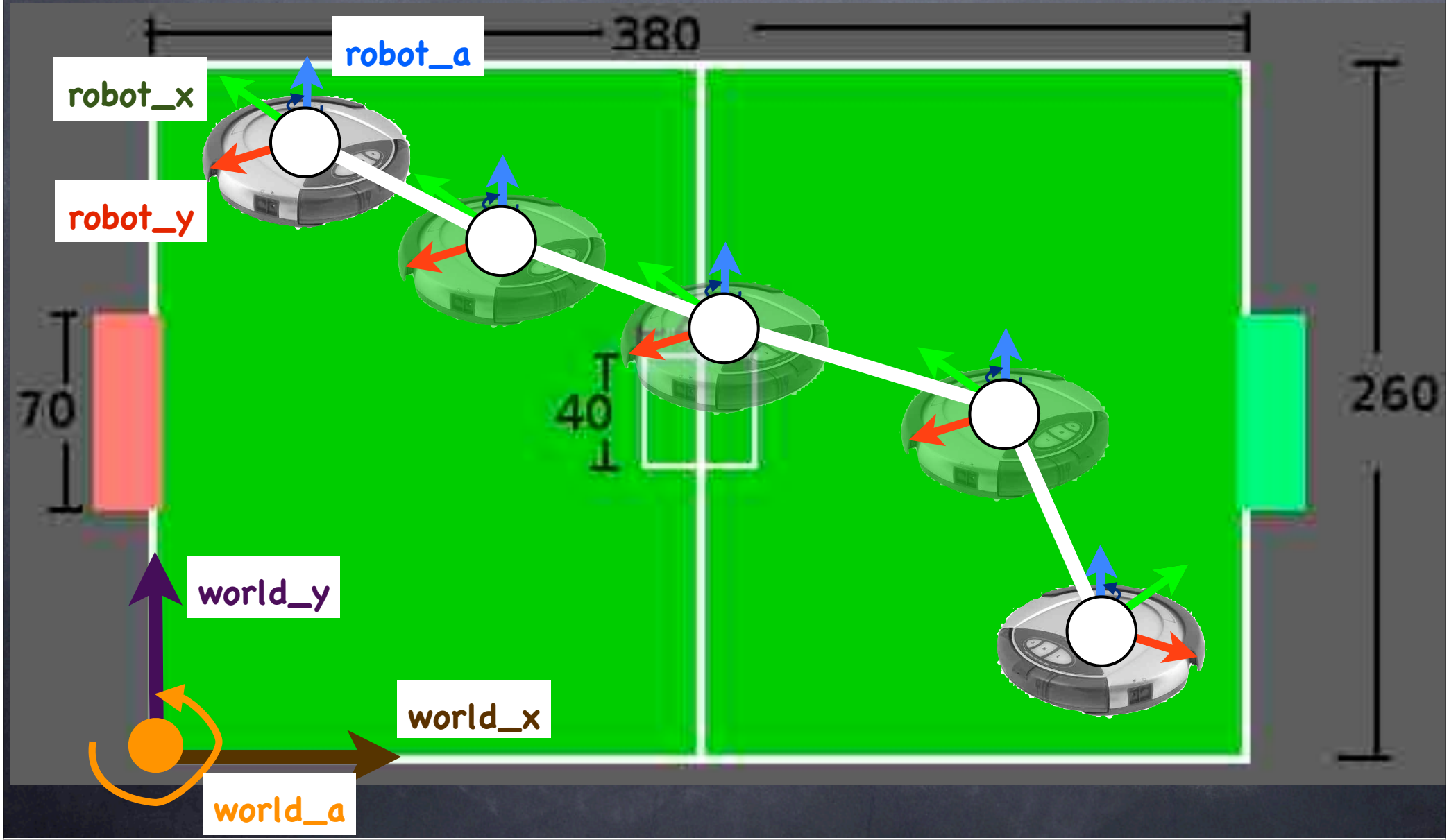
2) Define pose and controls?



1) path planning: sequence of intermediate poses

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

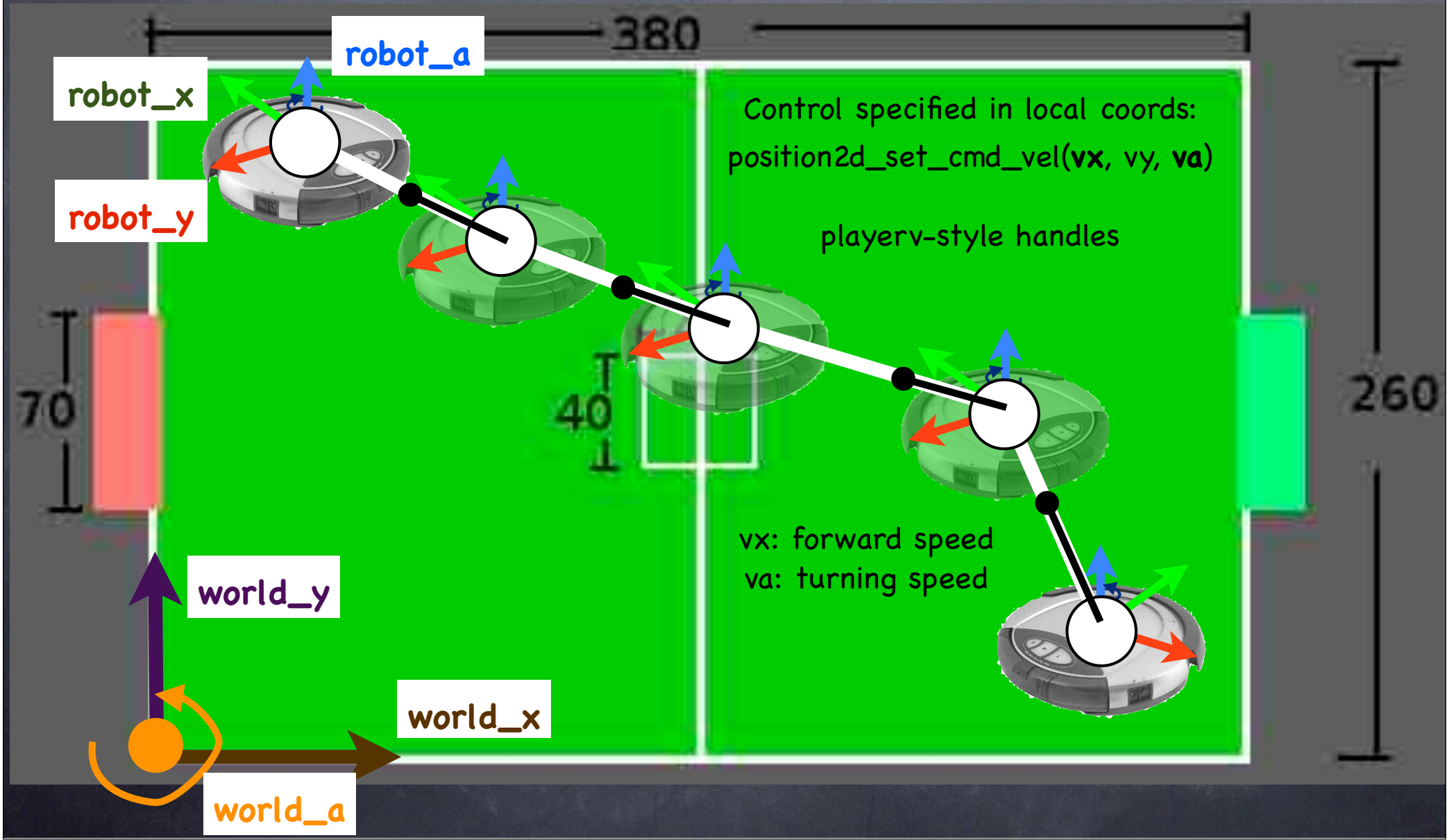
and controls?



1) path planning: sequence of intermediate poses

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

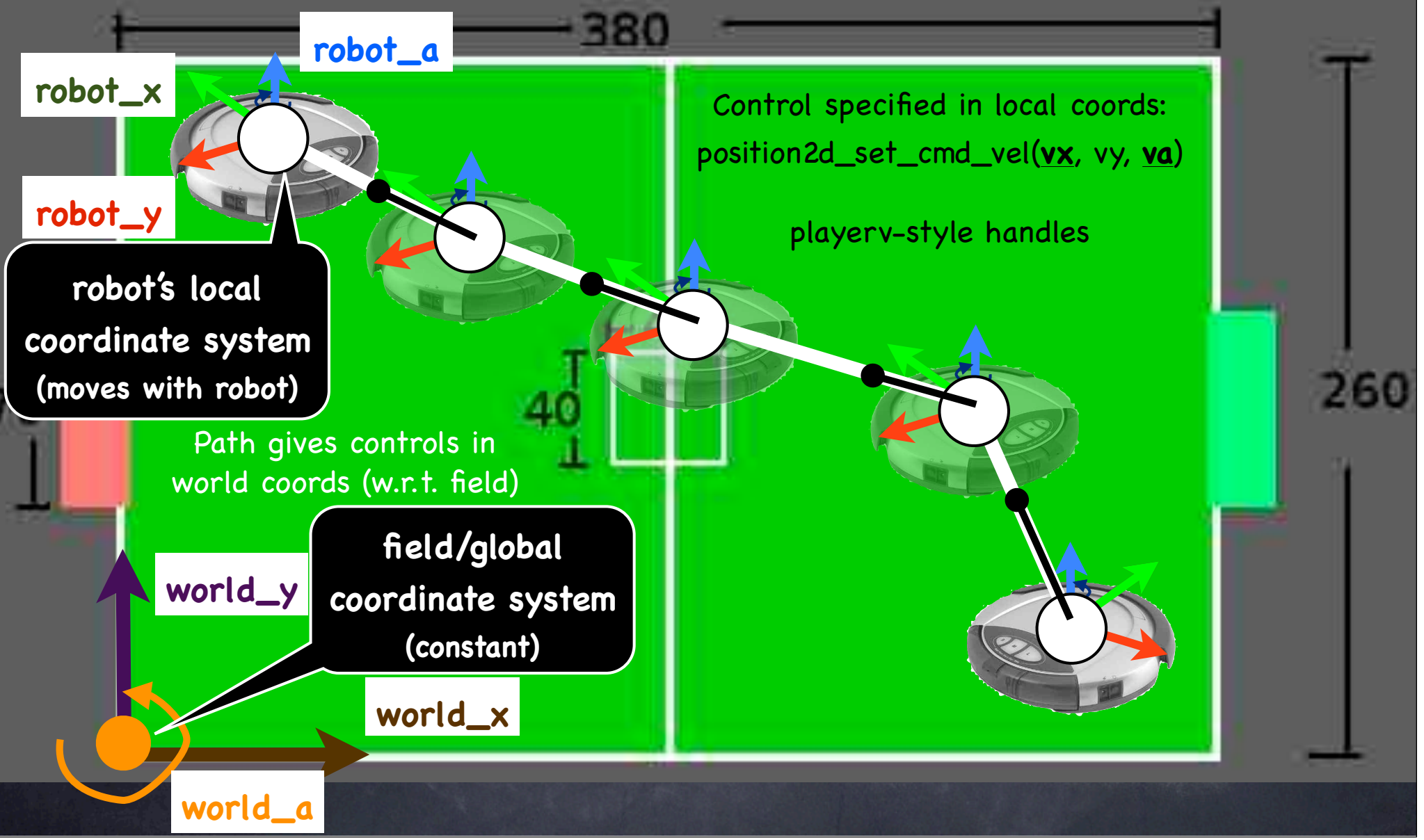
and controls?

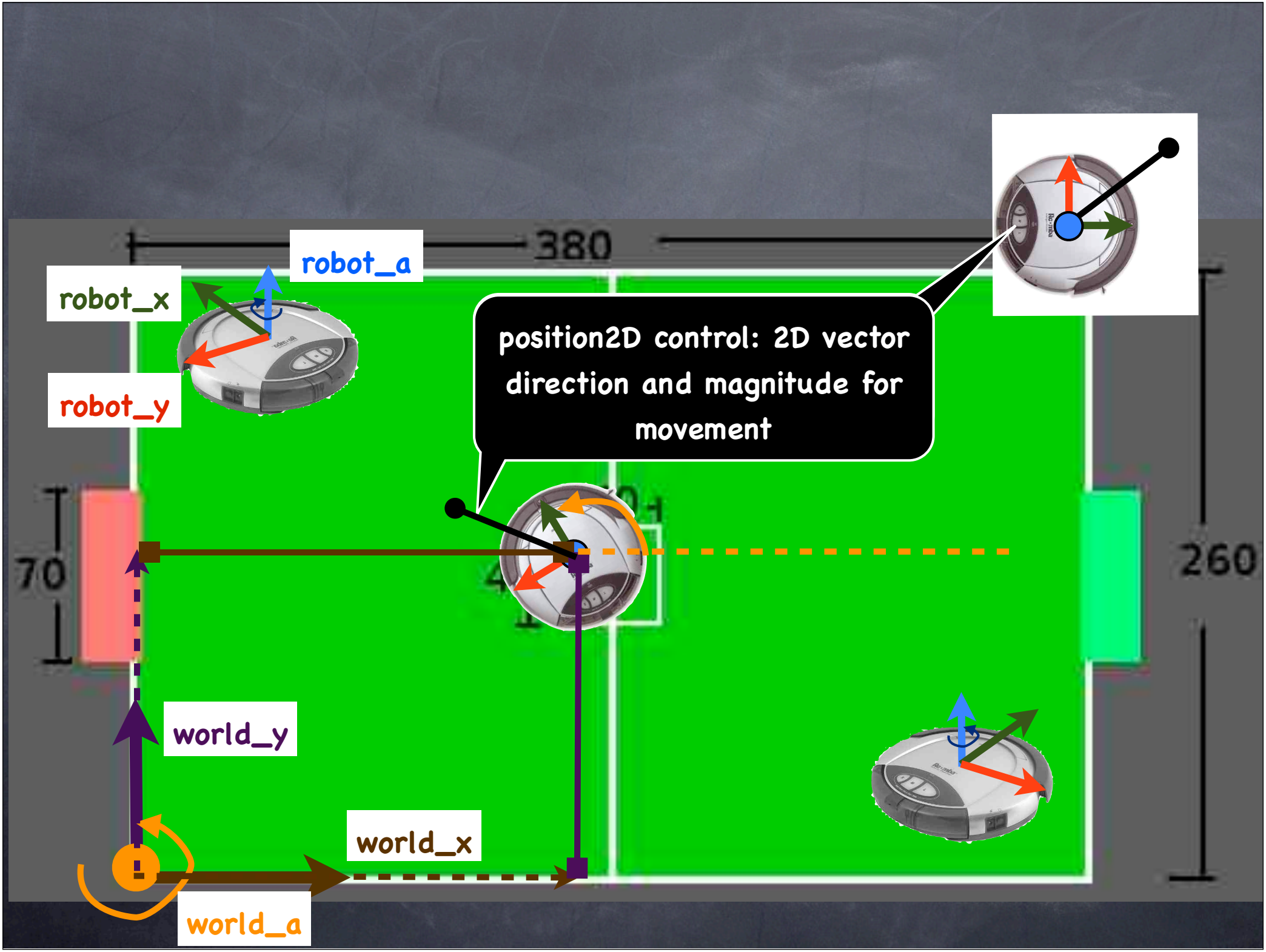


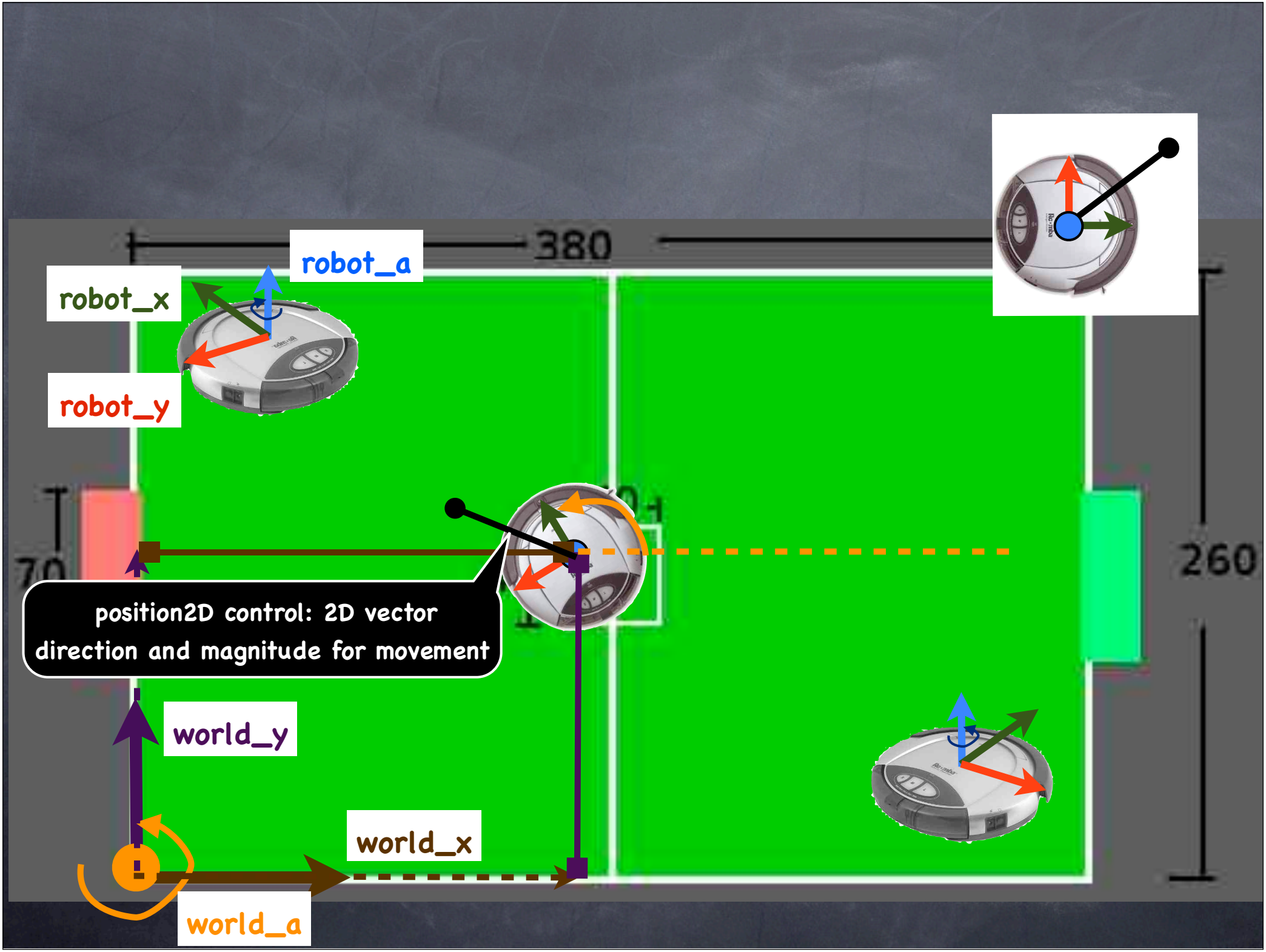
1) path planning: sequence of intermediate poses

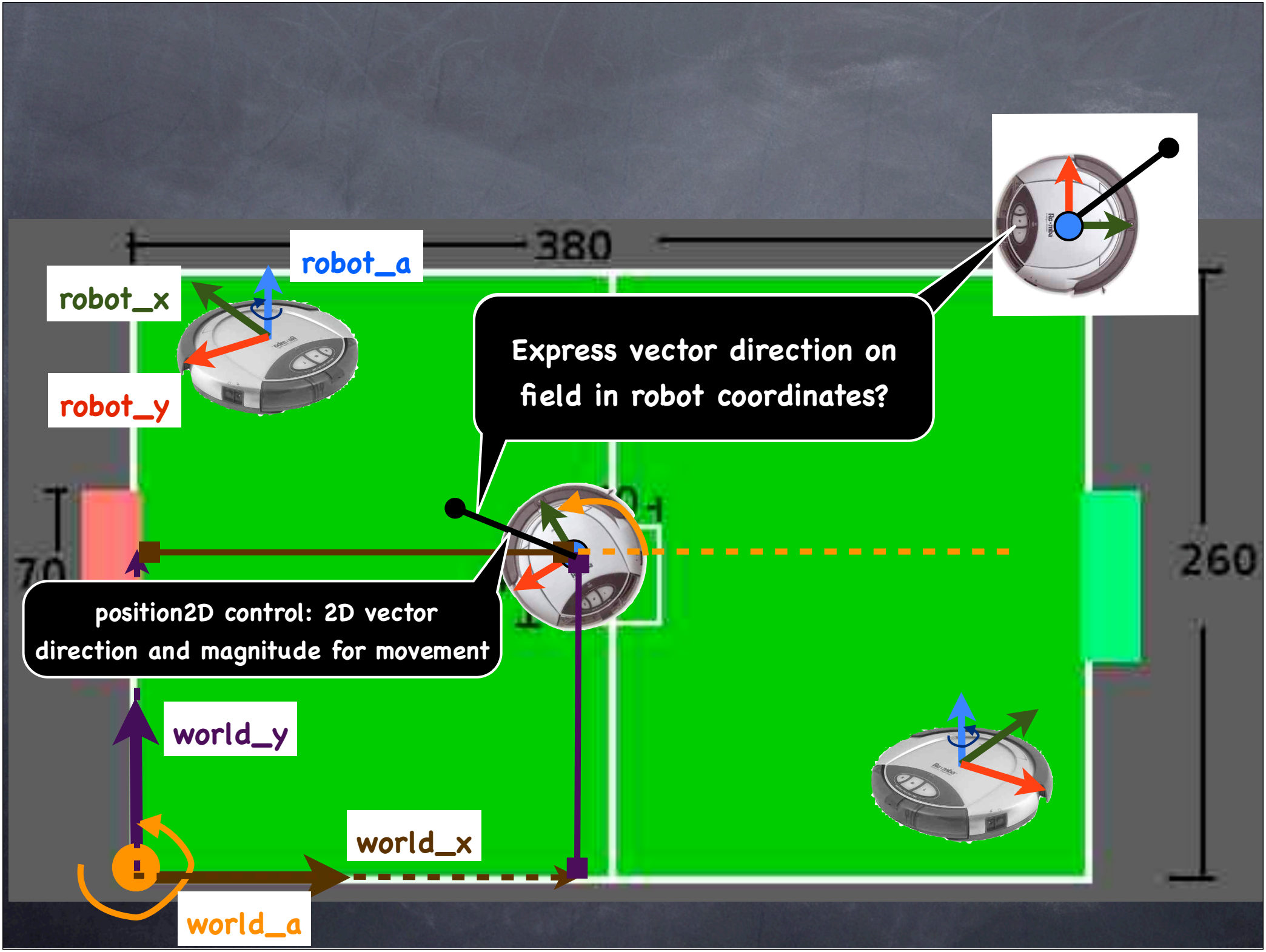
2) Define pose (3DOF position/orientation on field)

and controls (transform global control into robot coordinates)?

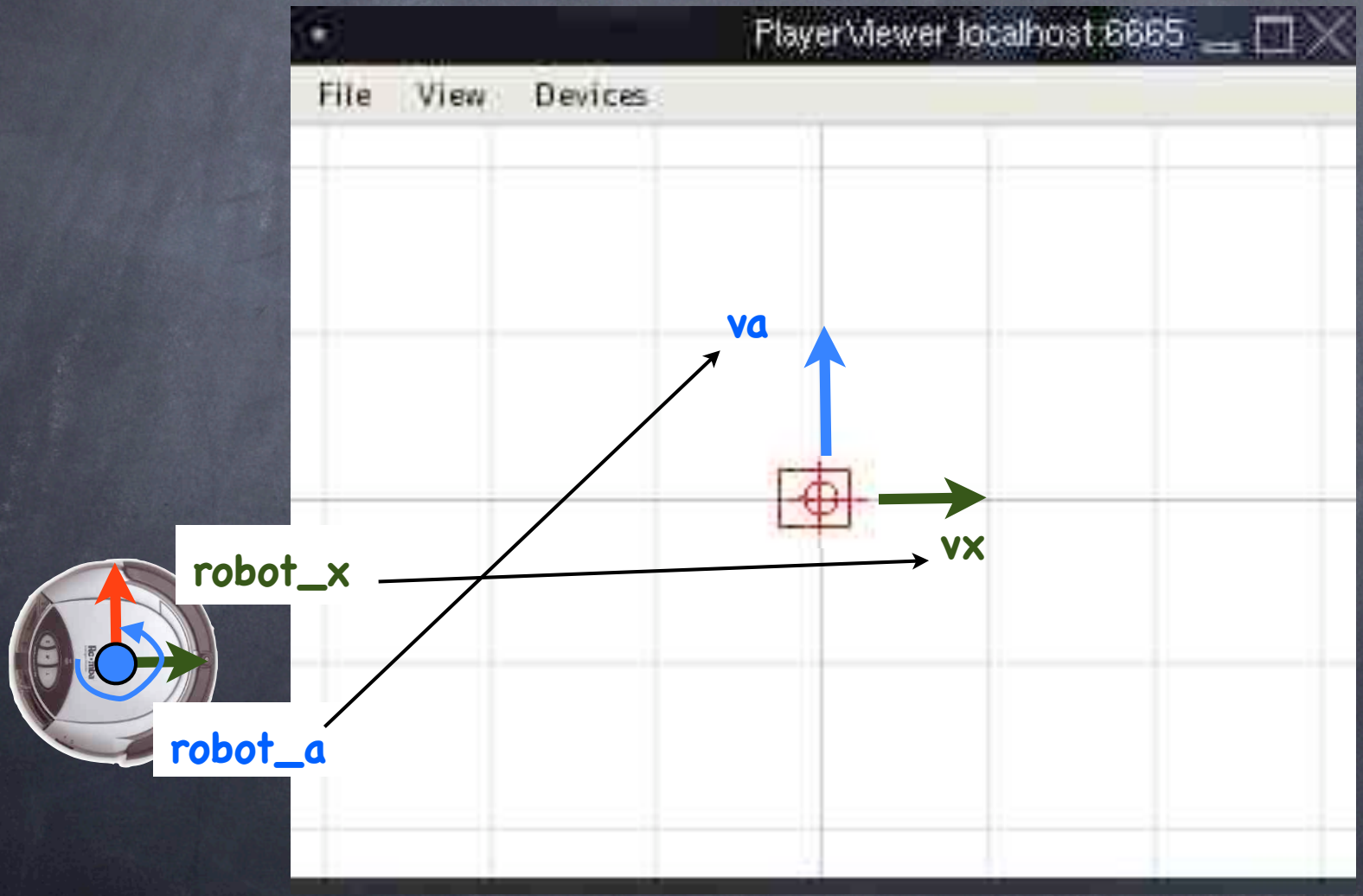



380
robot_a
robot_x
robot_y
position2D control: 2D vector direction and magnitude for movement
70
260
world_y
world_x
world_a

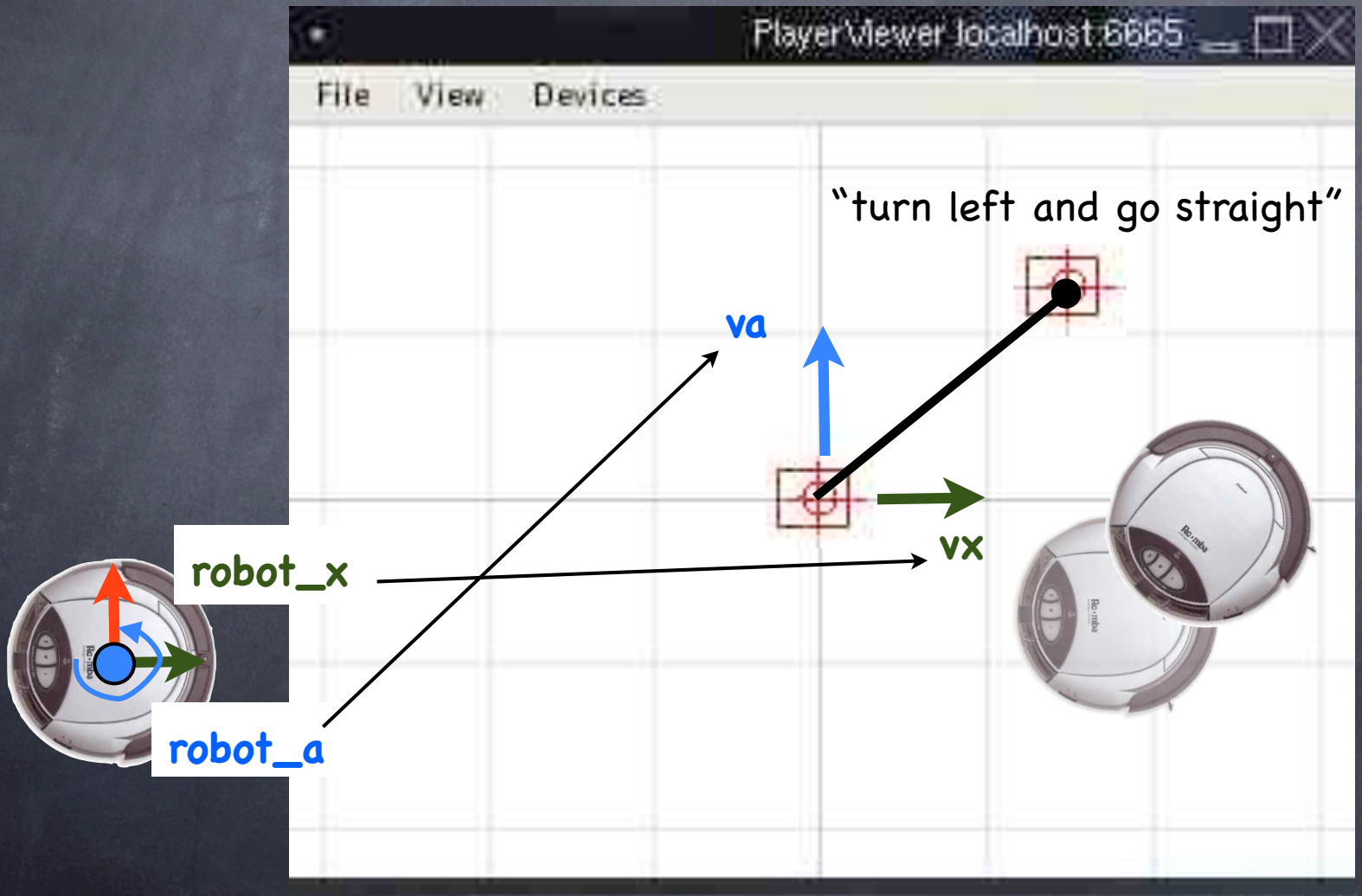

380
robot_a
robot_x
robot_y
260
position2D control: 2D vector
direction and magnitude for movement
world_y
world_x
world_a


380
robot_a
robot_x
robot_y
Express vector direction on field in robot coordinates?
260
70
position2D control: 2D vector direction and magnitude for movement
world_y
world_x
world_a

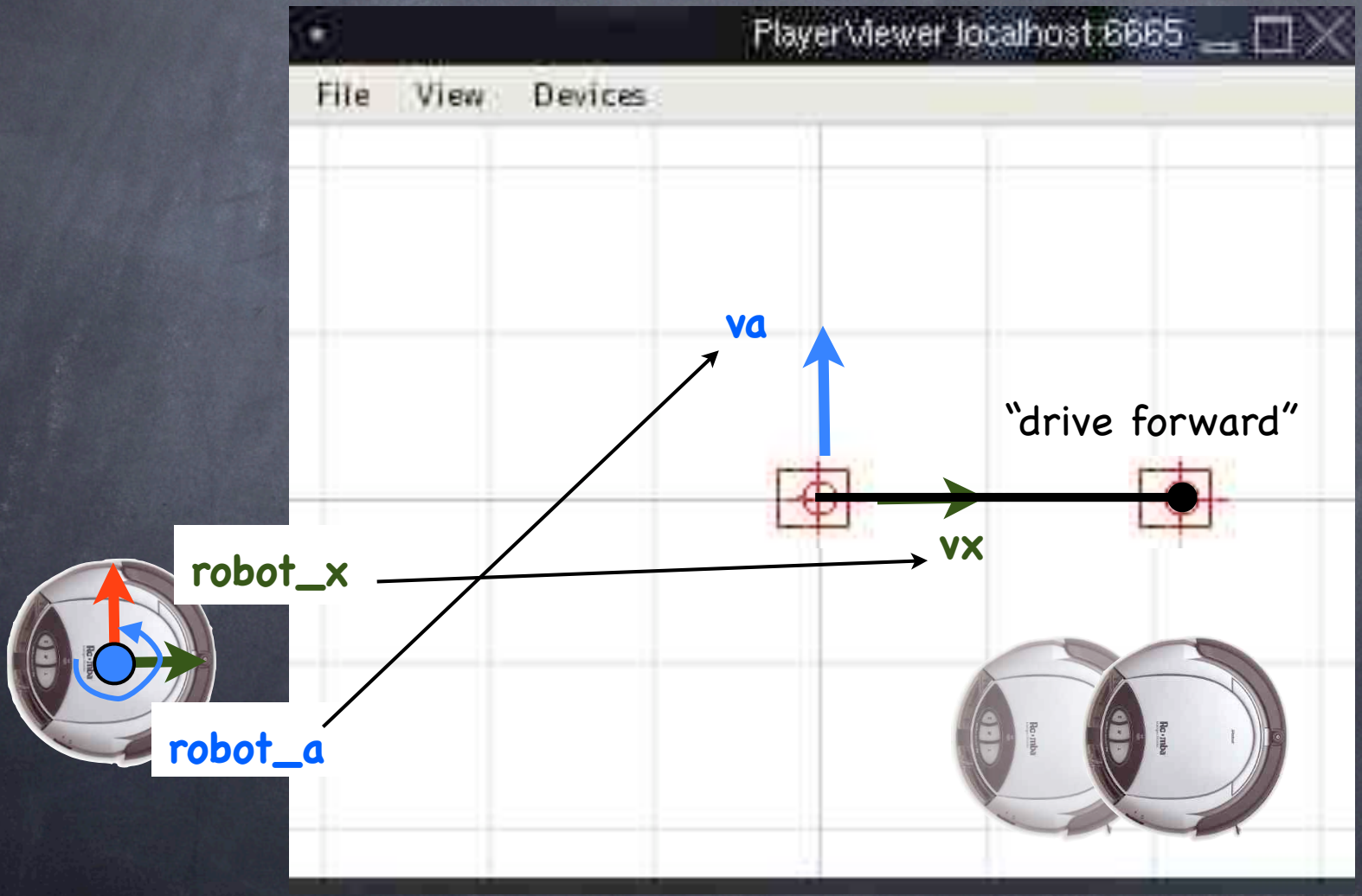
position2d in playerv



position2d in playerv

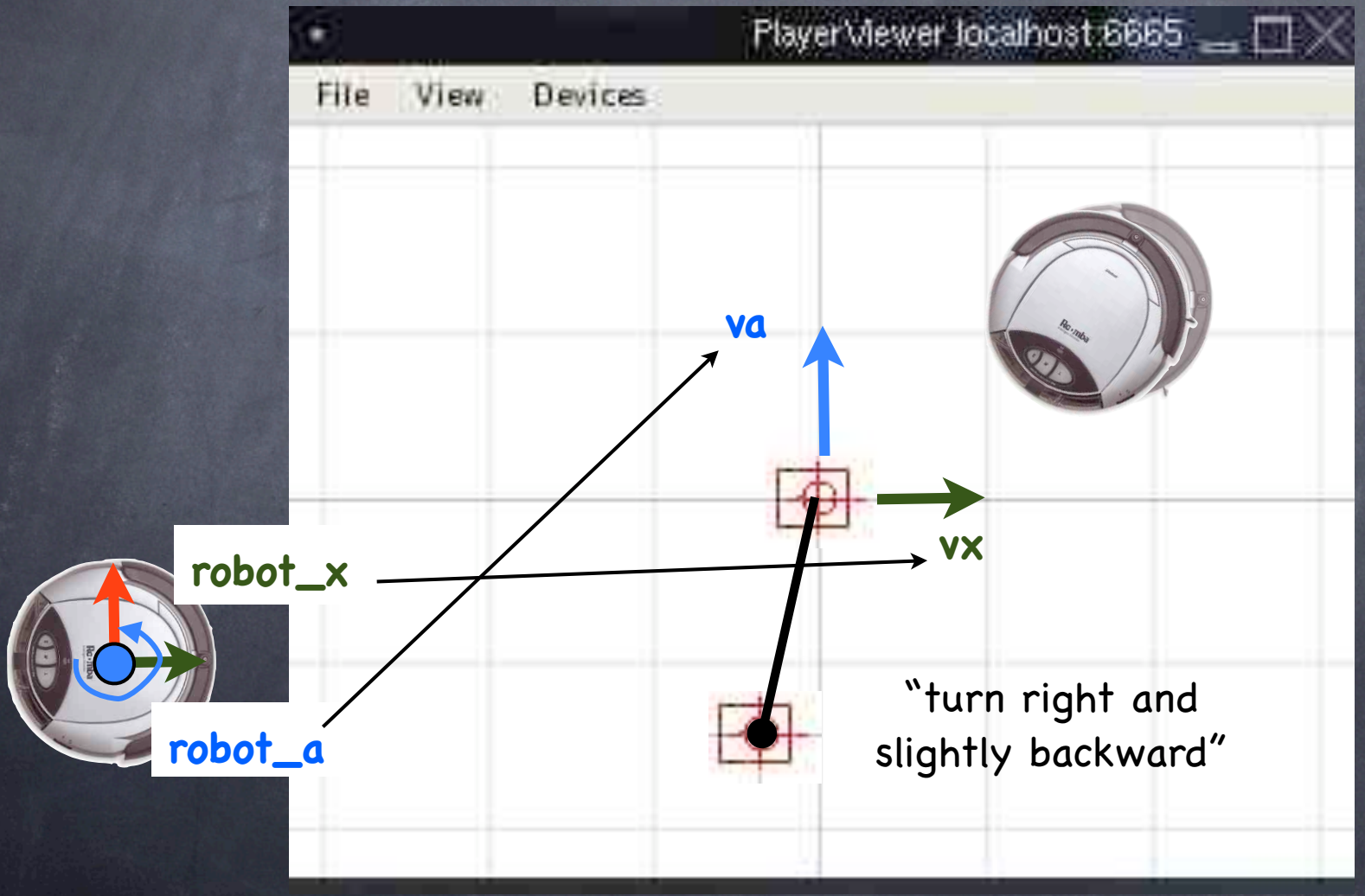


position2d in playerv



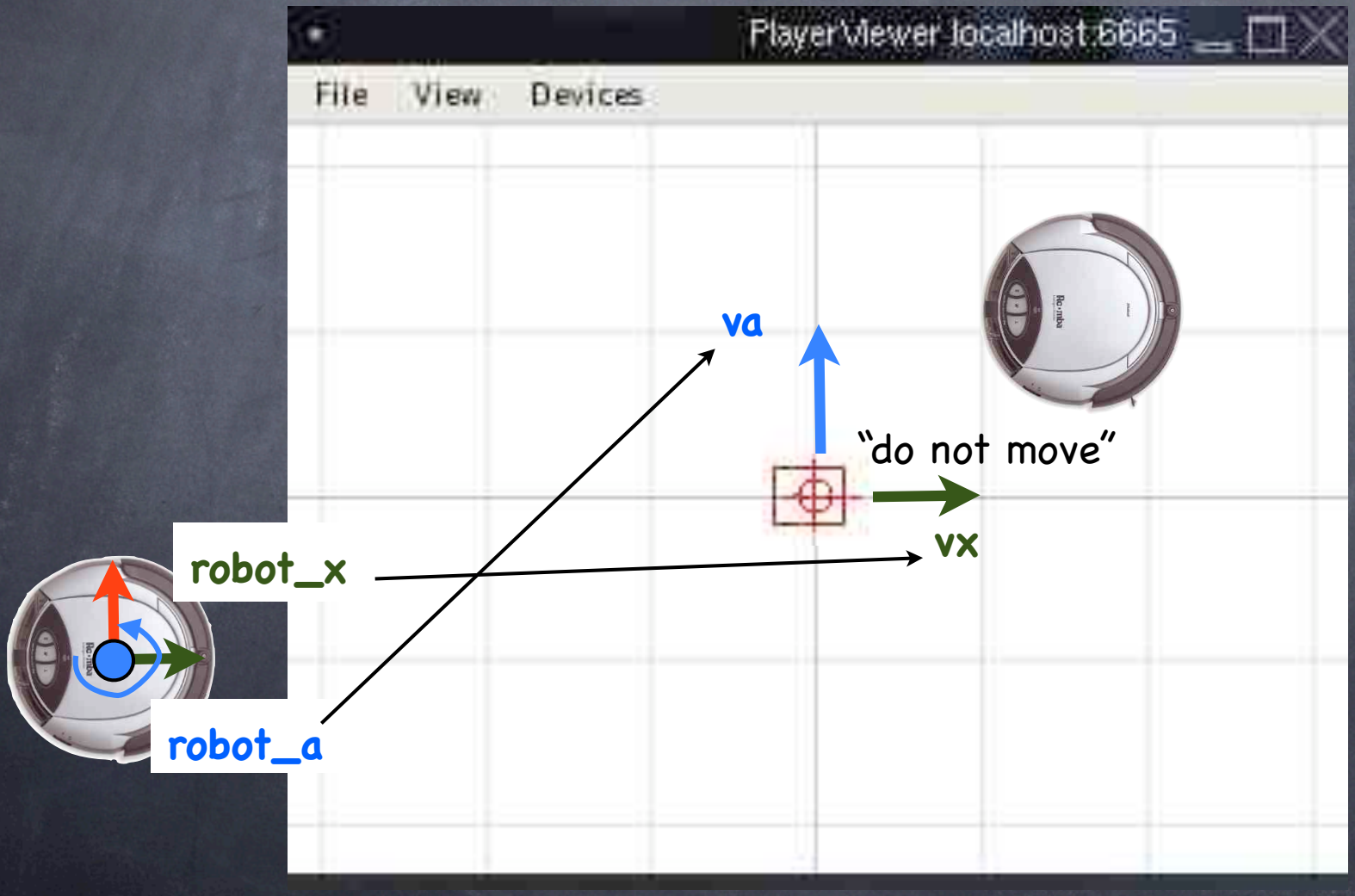
position2d in playerv

Can we be more precise?



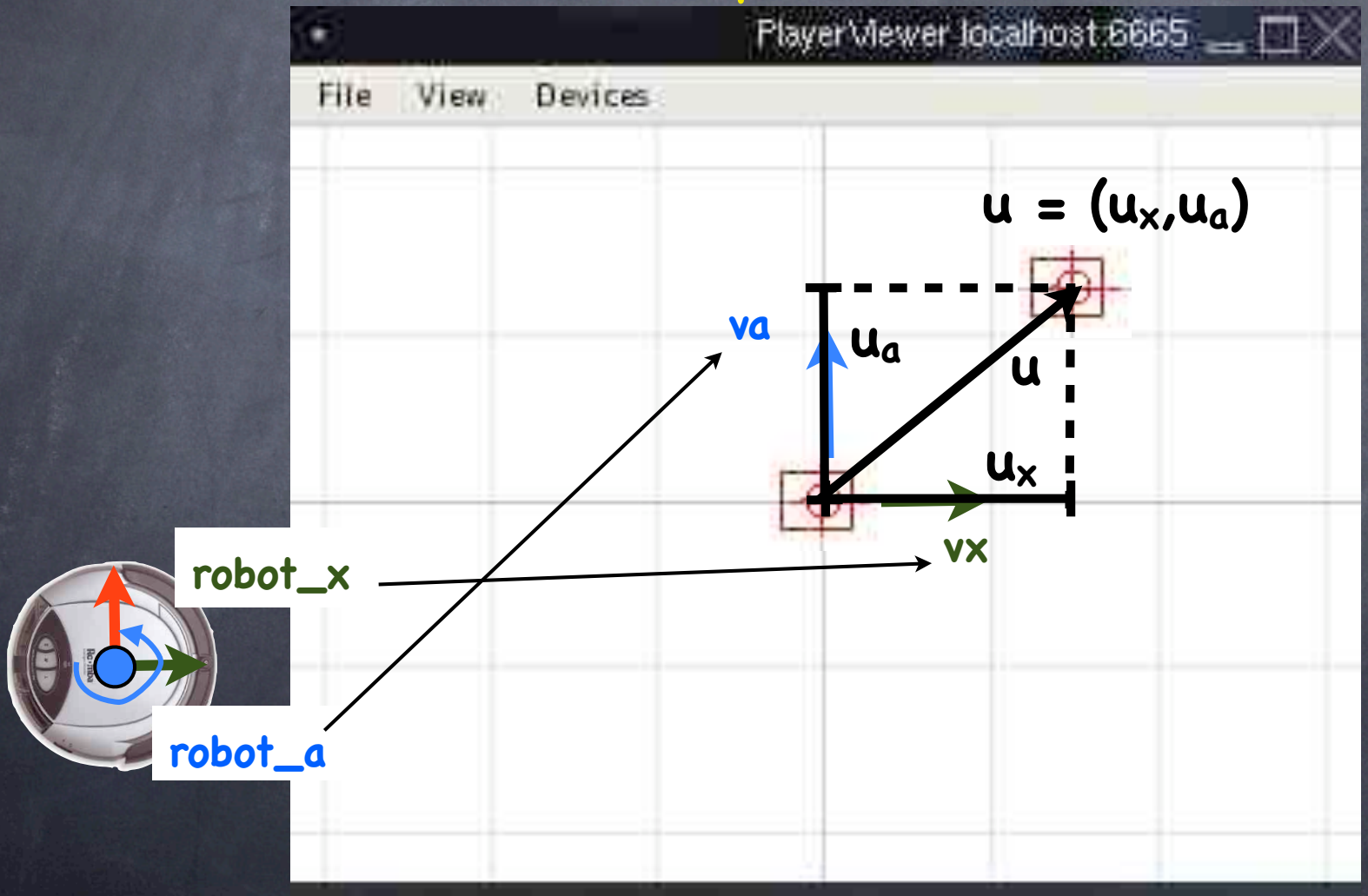
position2d in playerv

Can we be more precise?



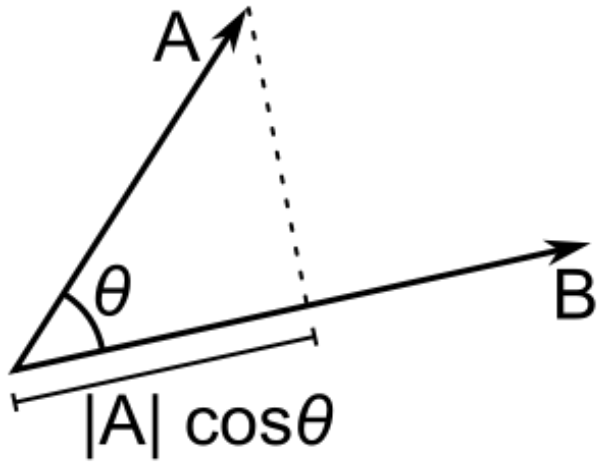
position2d in playerv

Note: u_a is dot product of u and va



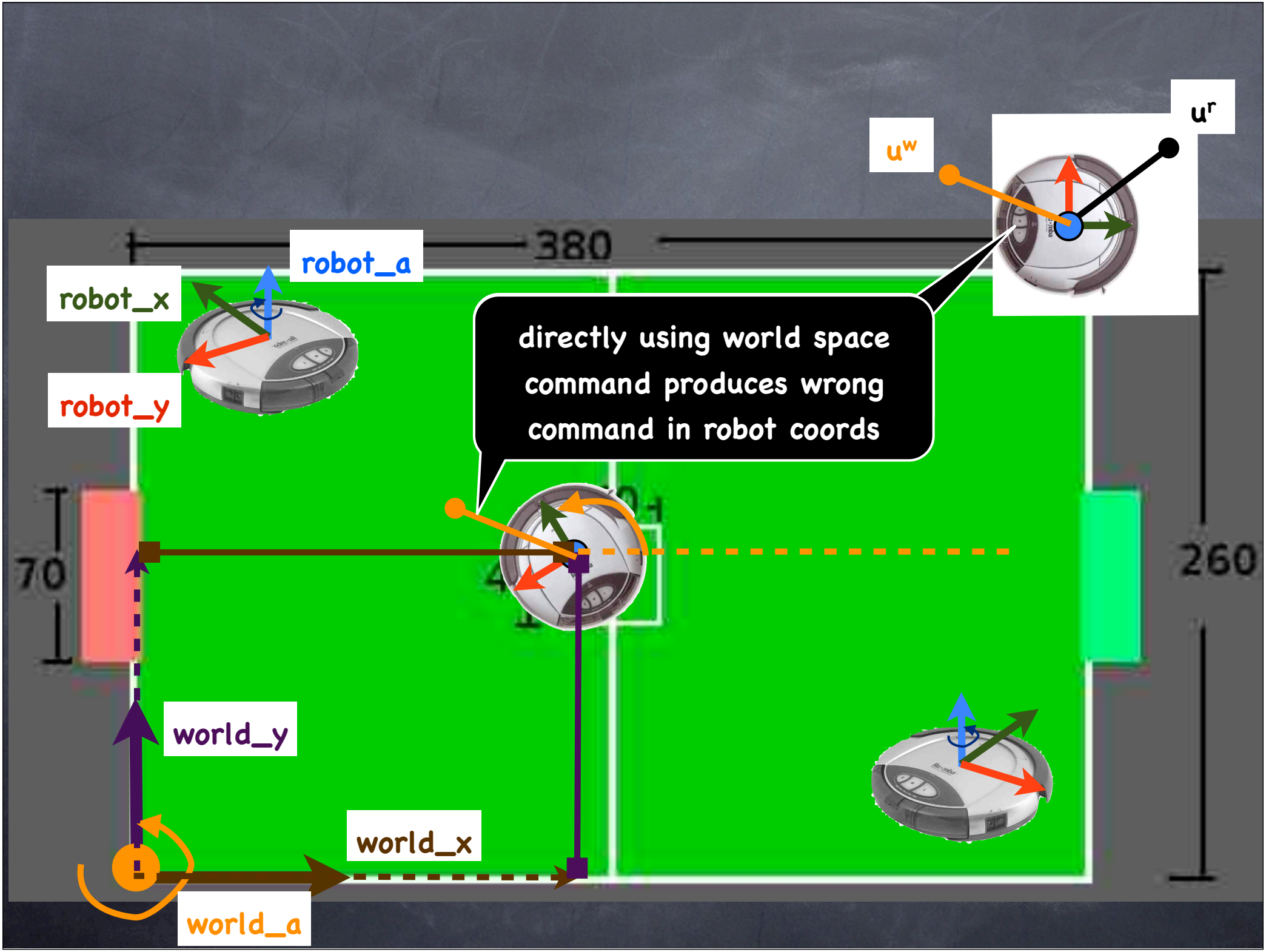
Dot Product

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



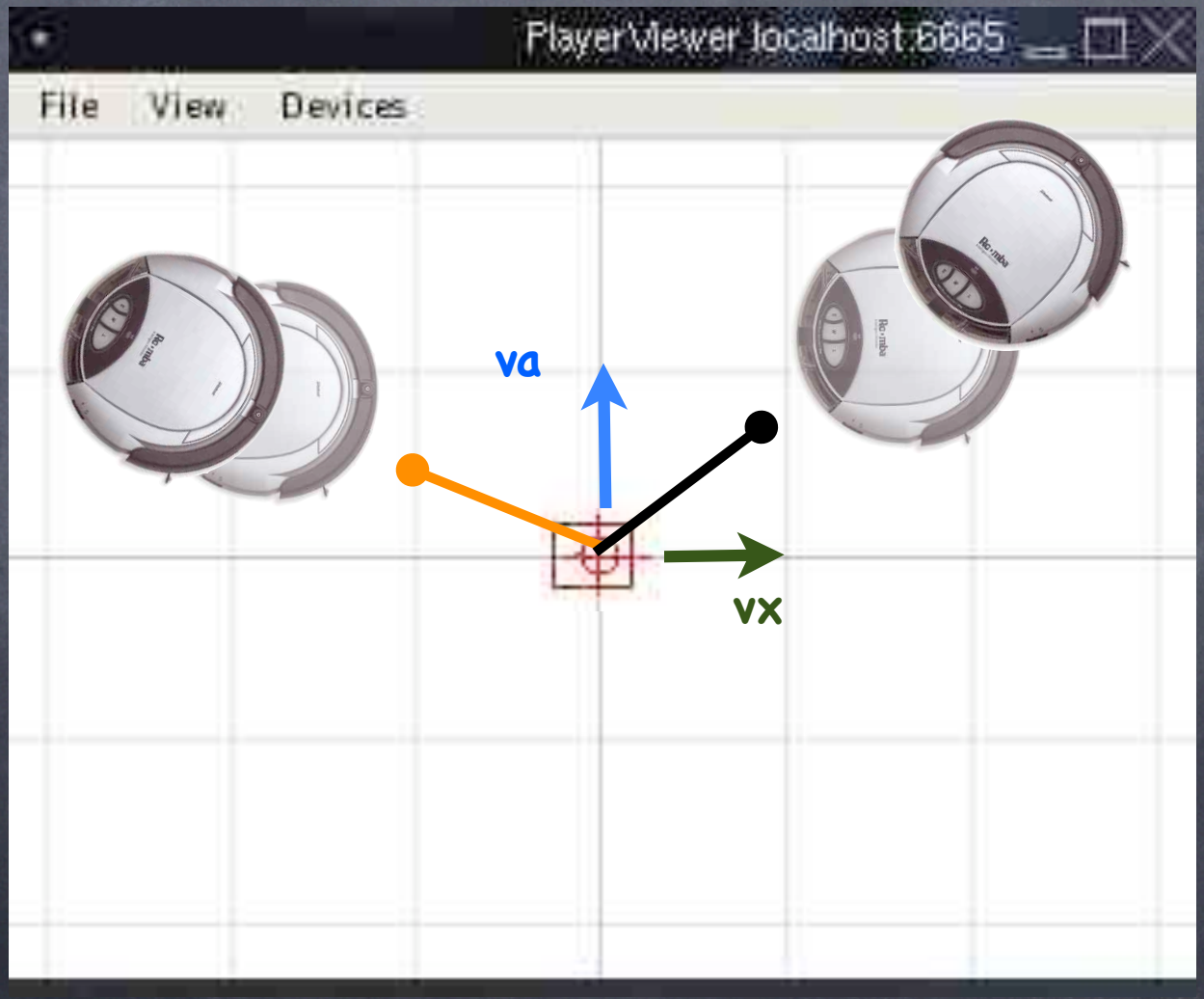
- If one is a unit vector (length 1), the dot product is the projection onto this vector
- Related to angle between vectors

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

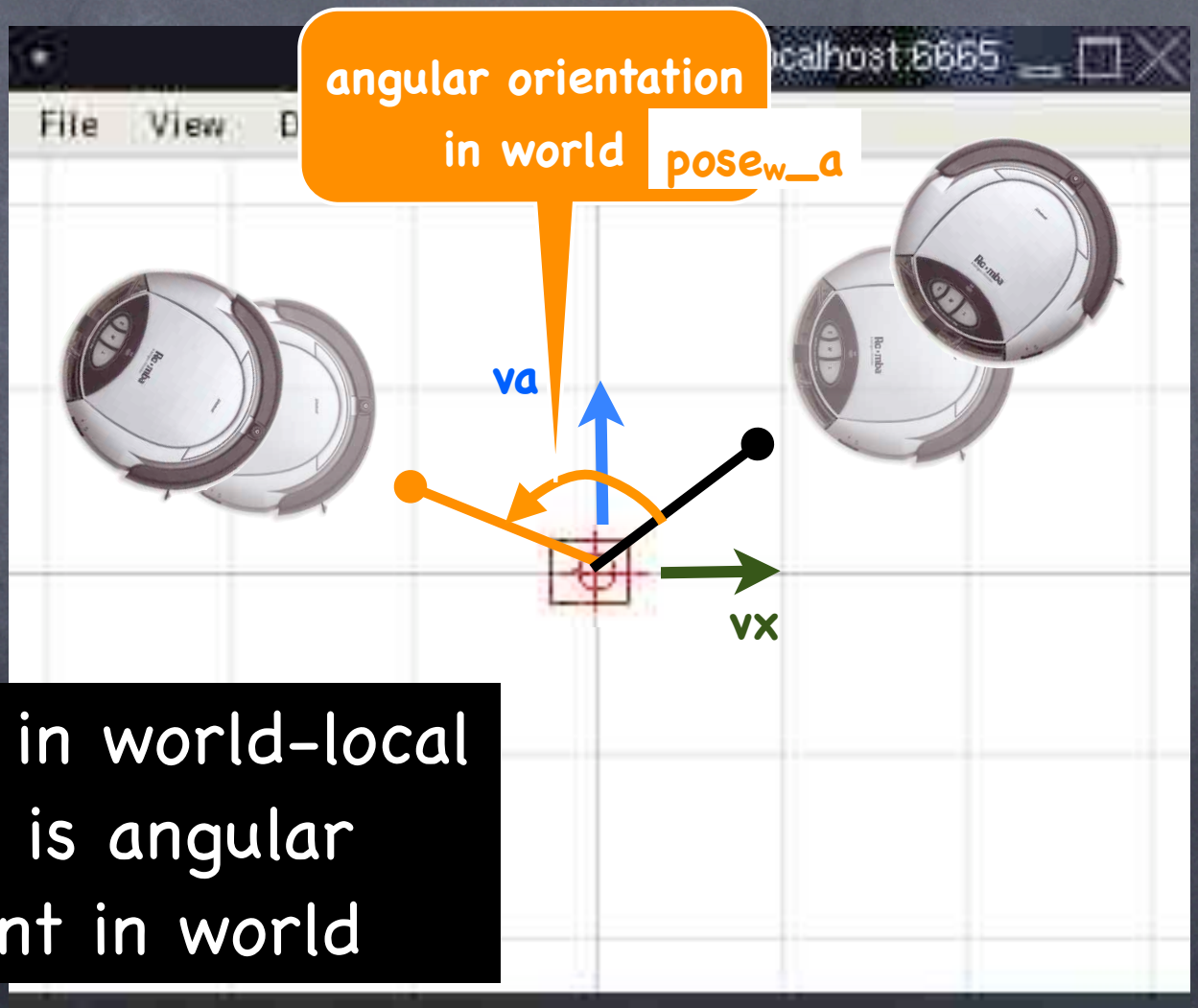

380
robot_a
robot_x
robot_y
directly using world space
command produces wrong
command in robot coords
u^w
u^r
70
260
world_y
world_x
world_a

position2d in playerv

How can we transform control vector into robot coords?

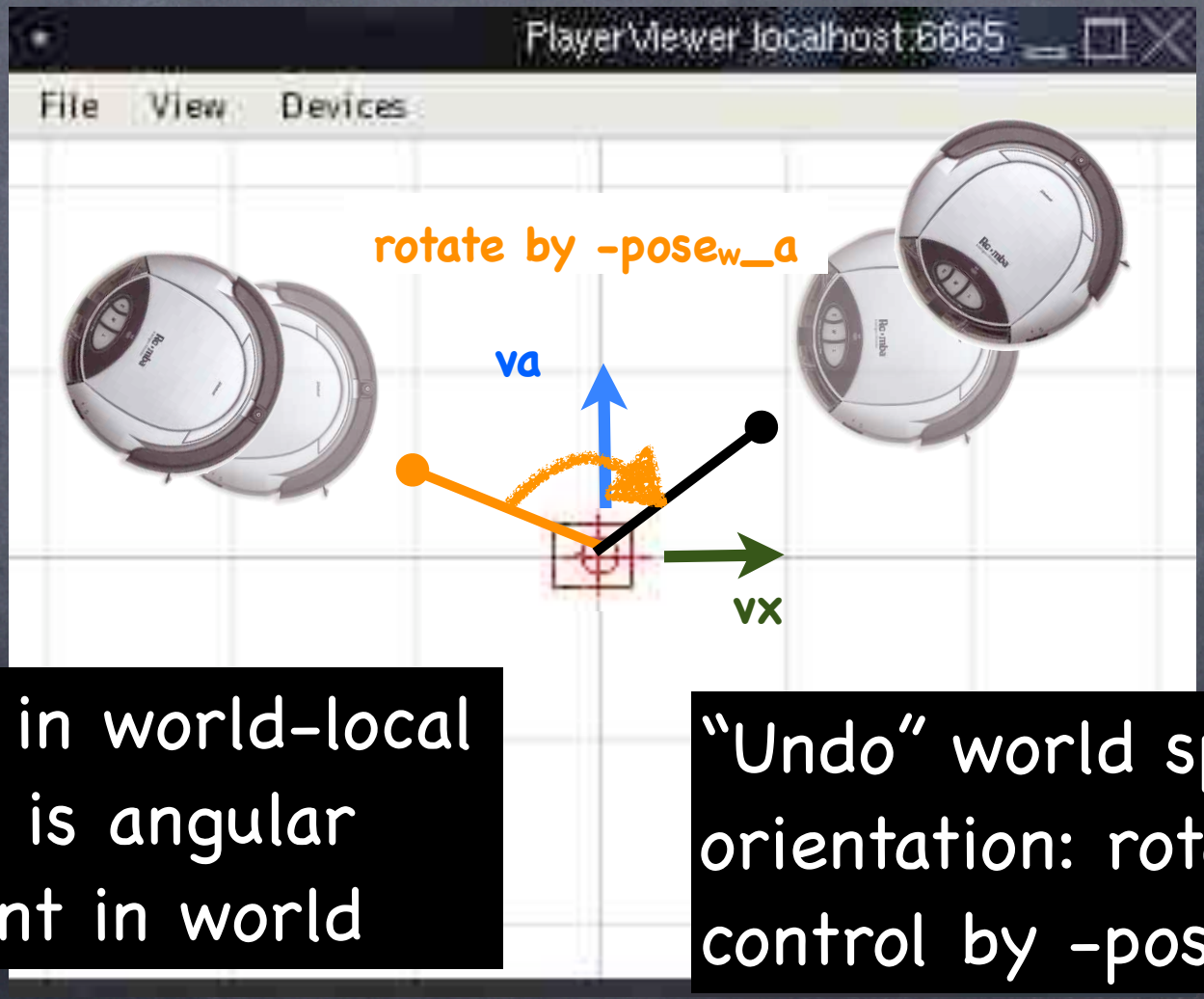


position2d in playerv



Difference in world-local orientation is angular displacement in world

position2d in playerv

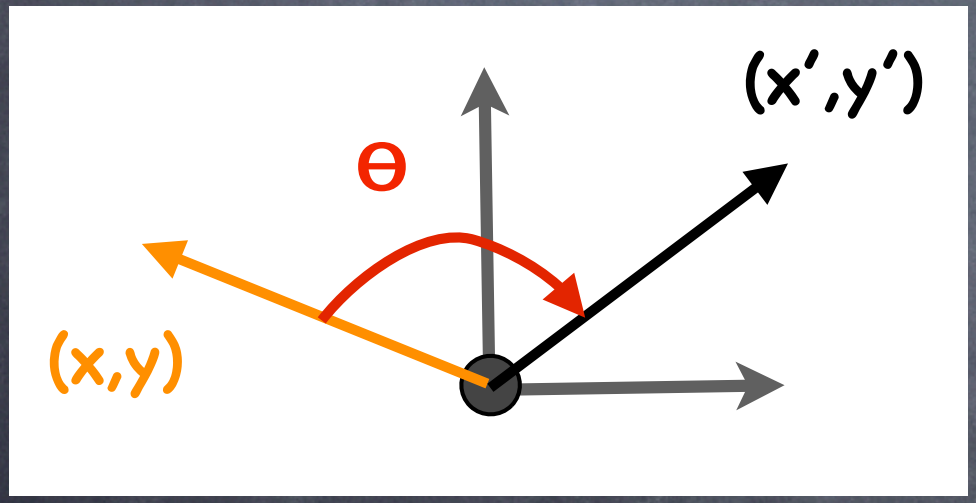


Difference in world-local orientation is angular displacement in world

"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

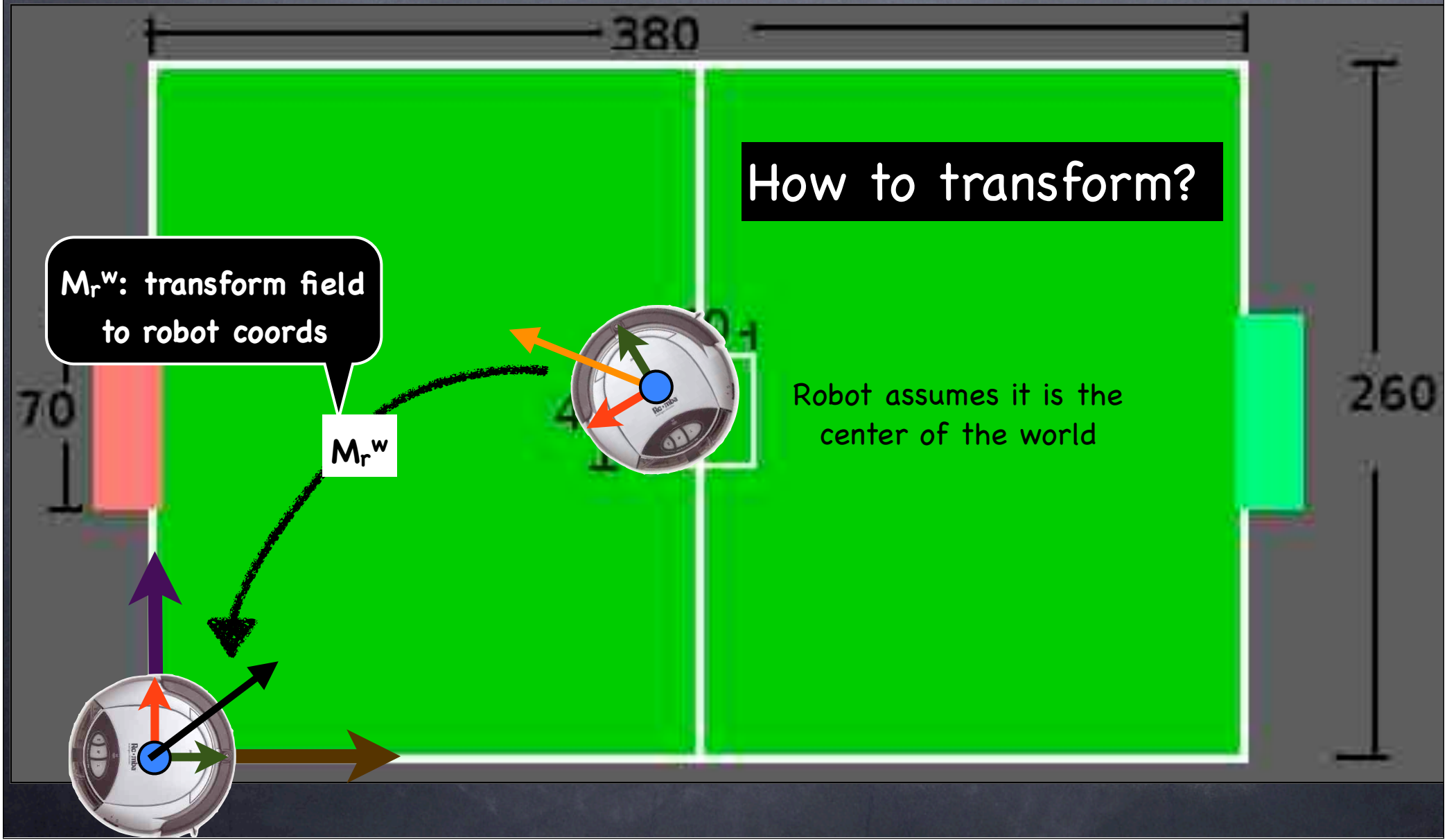
Matrix multiplication

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

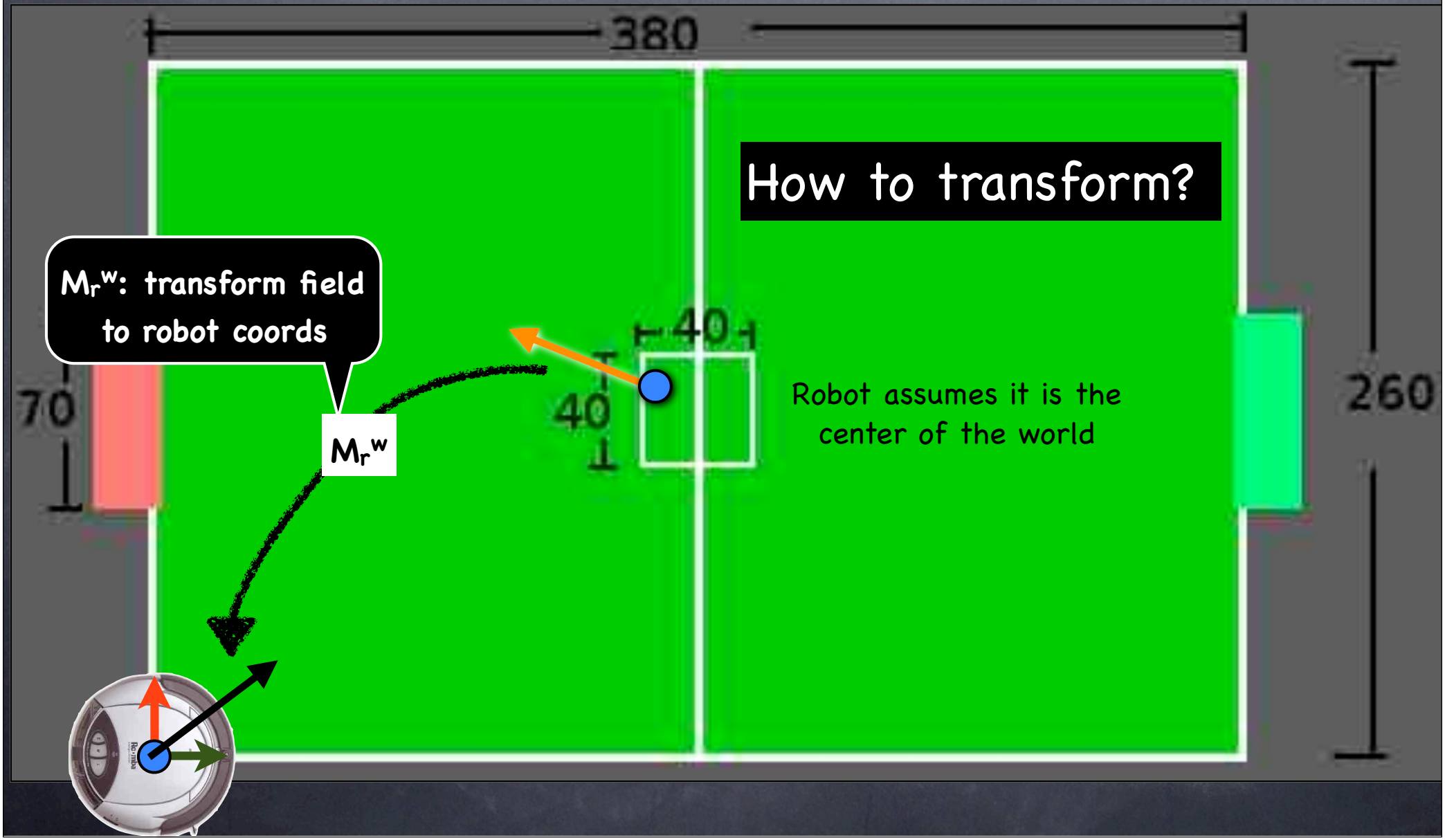
$$x_{3,4} = (1,2,3,4) \cdot (a,b,c,d)$$

$$= 1 \times a + 2 \times b + 3 \times c + 4 \times d$$

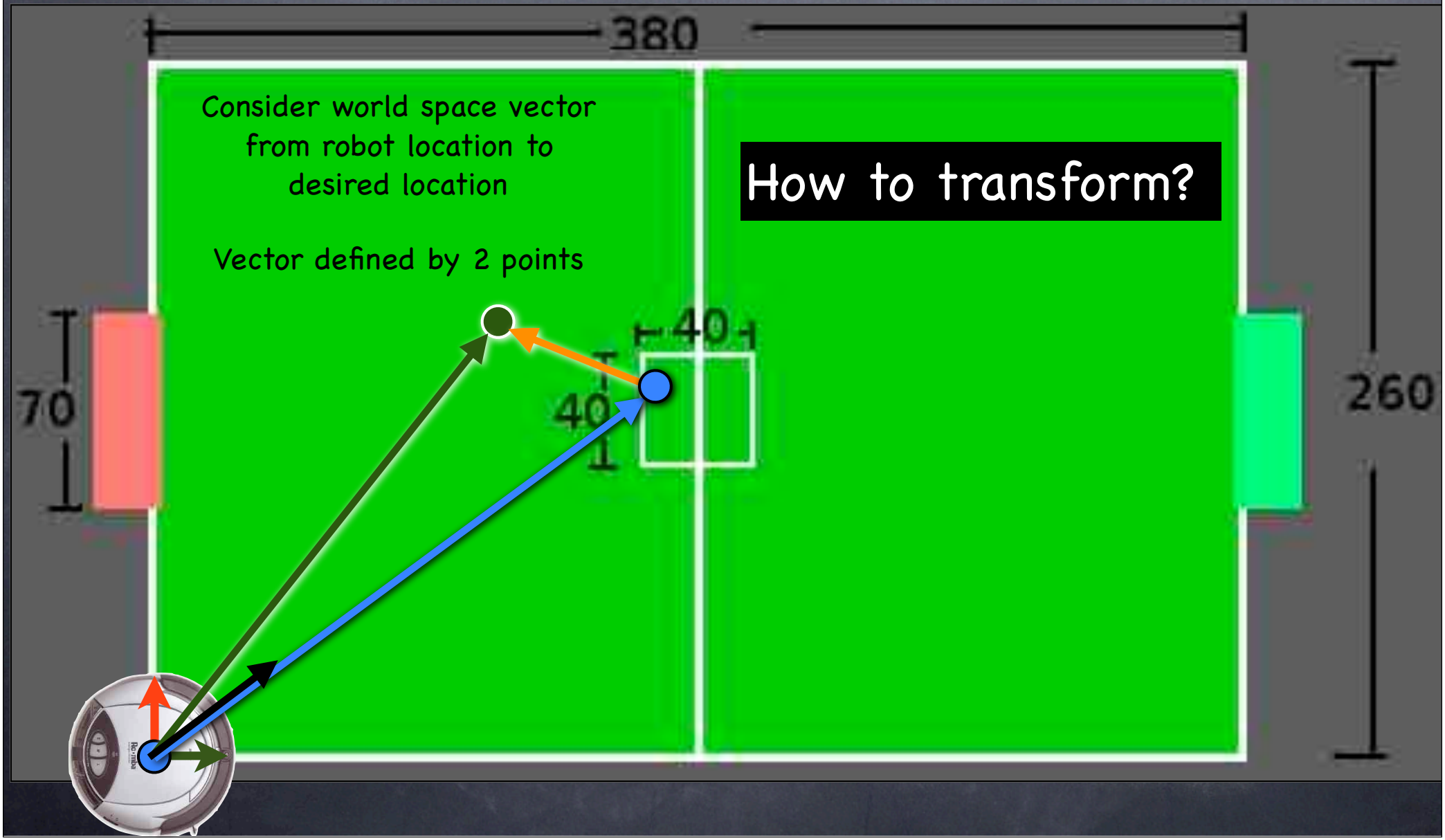
More generally, we should transform world space vector into robot local coordinates to yield command vector



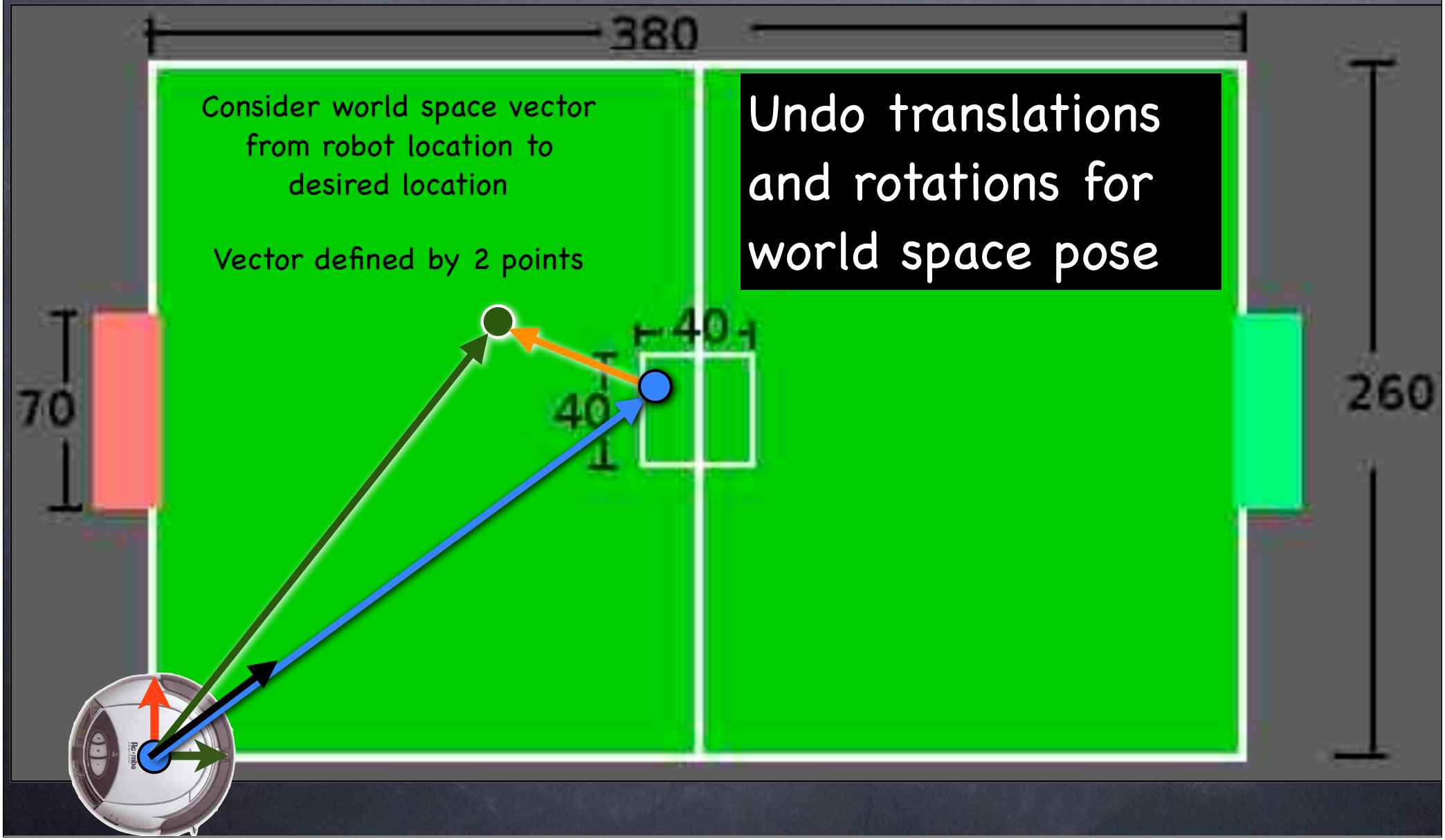
More generally, we should transform world space vector into robot local coordinates to yield command vector



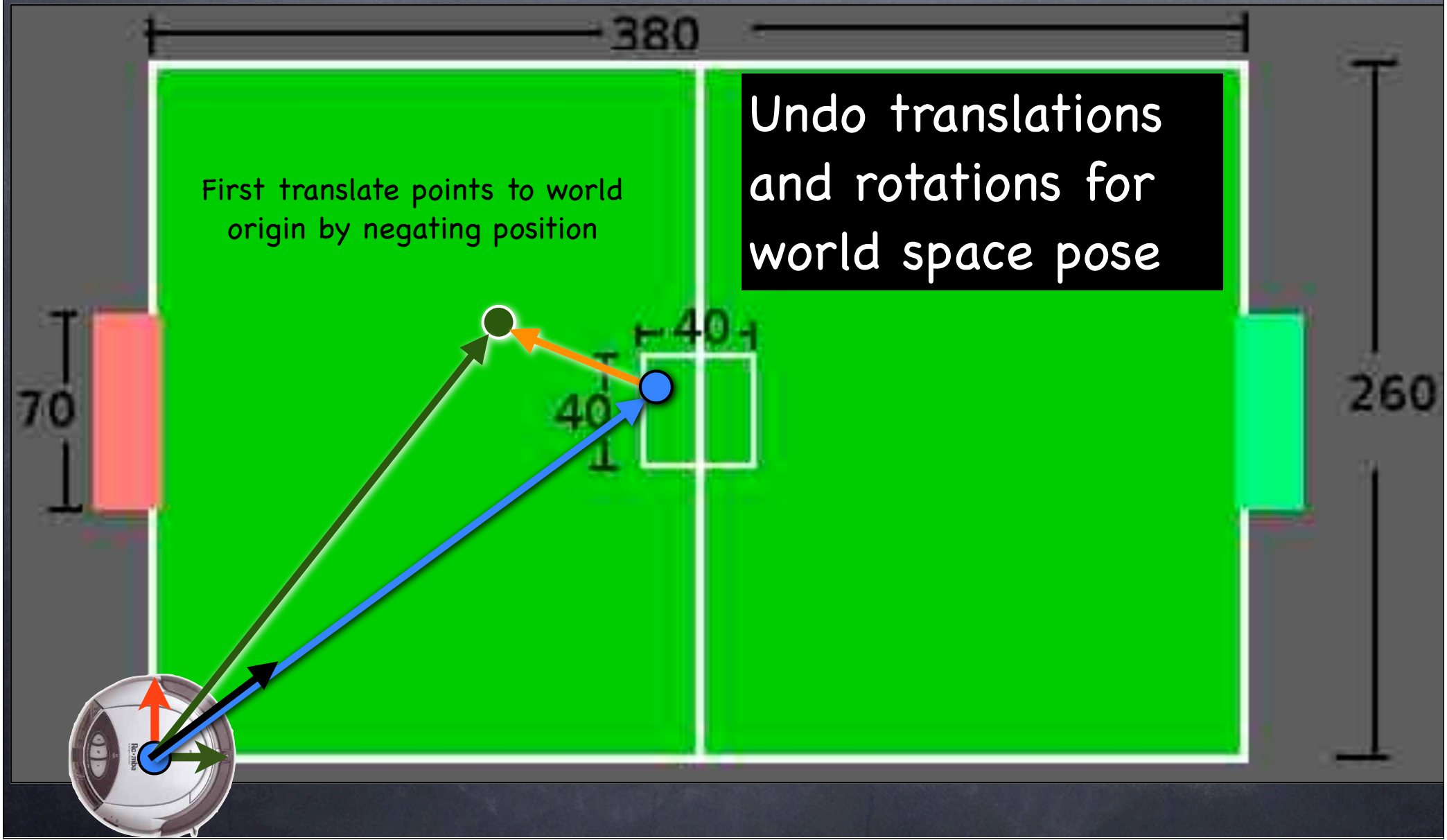
More generally, we should transform world space vector into robot local coordinates to yield command vector

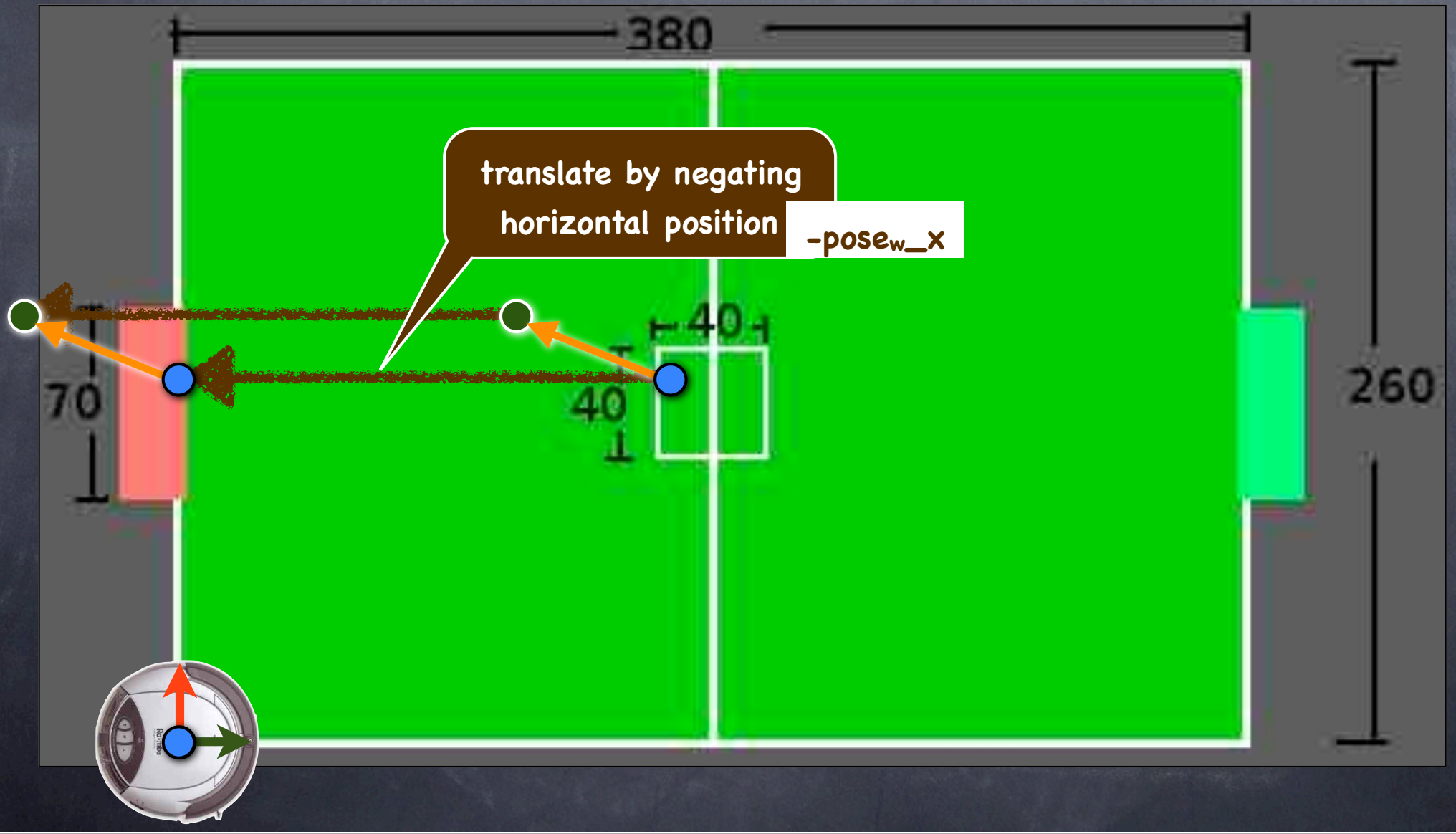


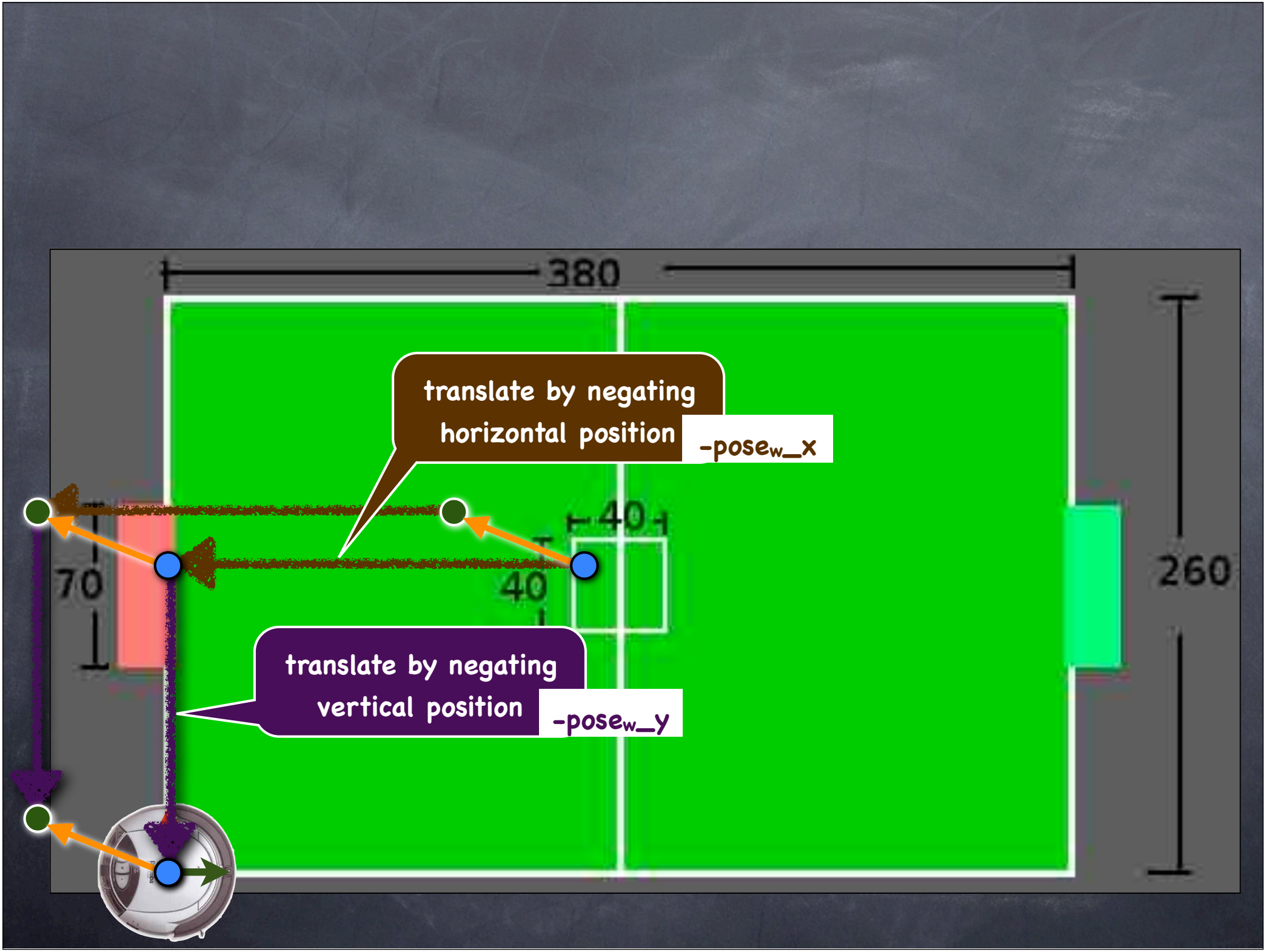
More generally, we should transform world space vector into robot local coordinates to yield command vector



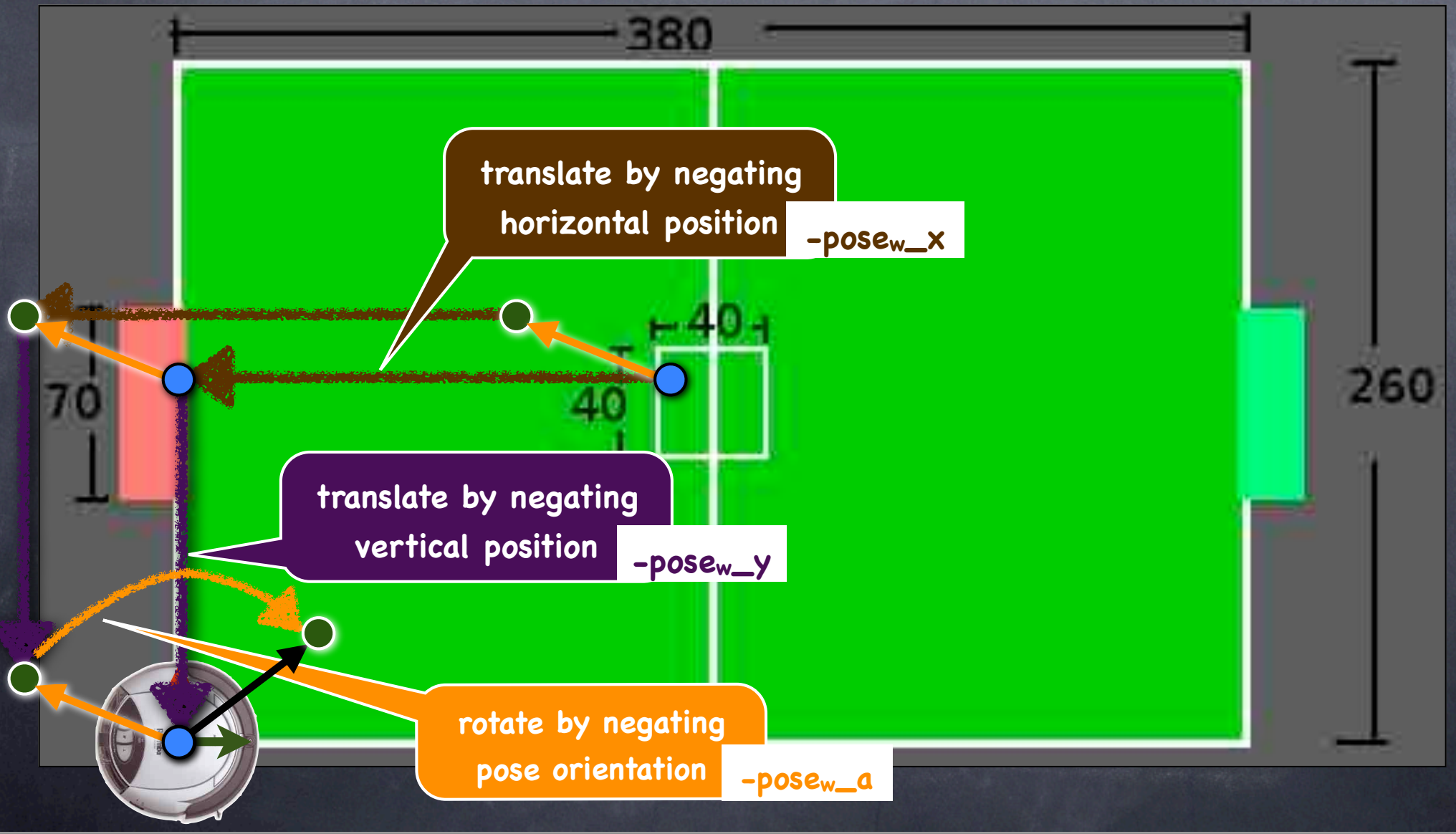
More generally, we should transform world space vector into robot local coordinates to yield command vector



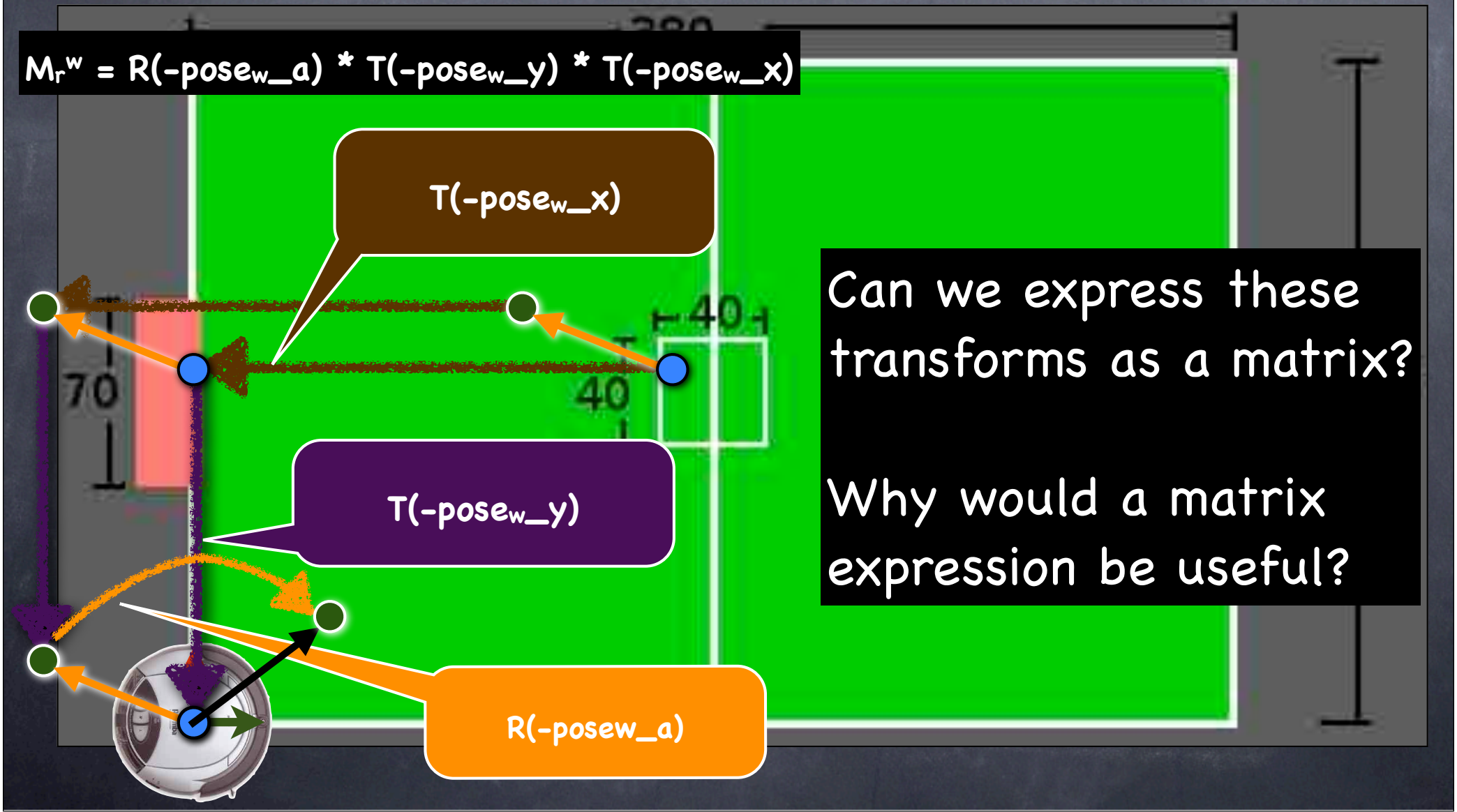

380
260
70
40
40
translate by negating
horizontal position
-pose_w_x


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

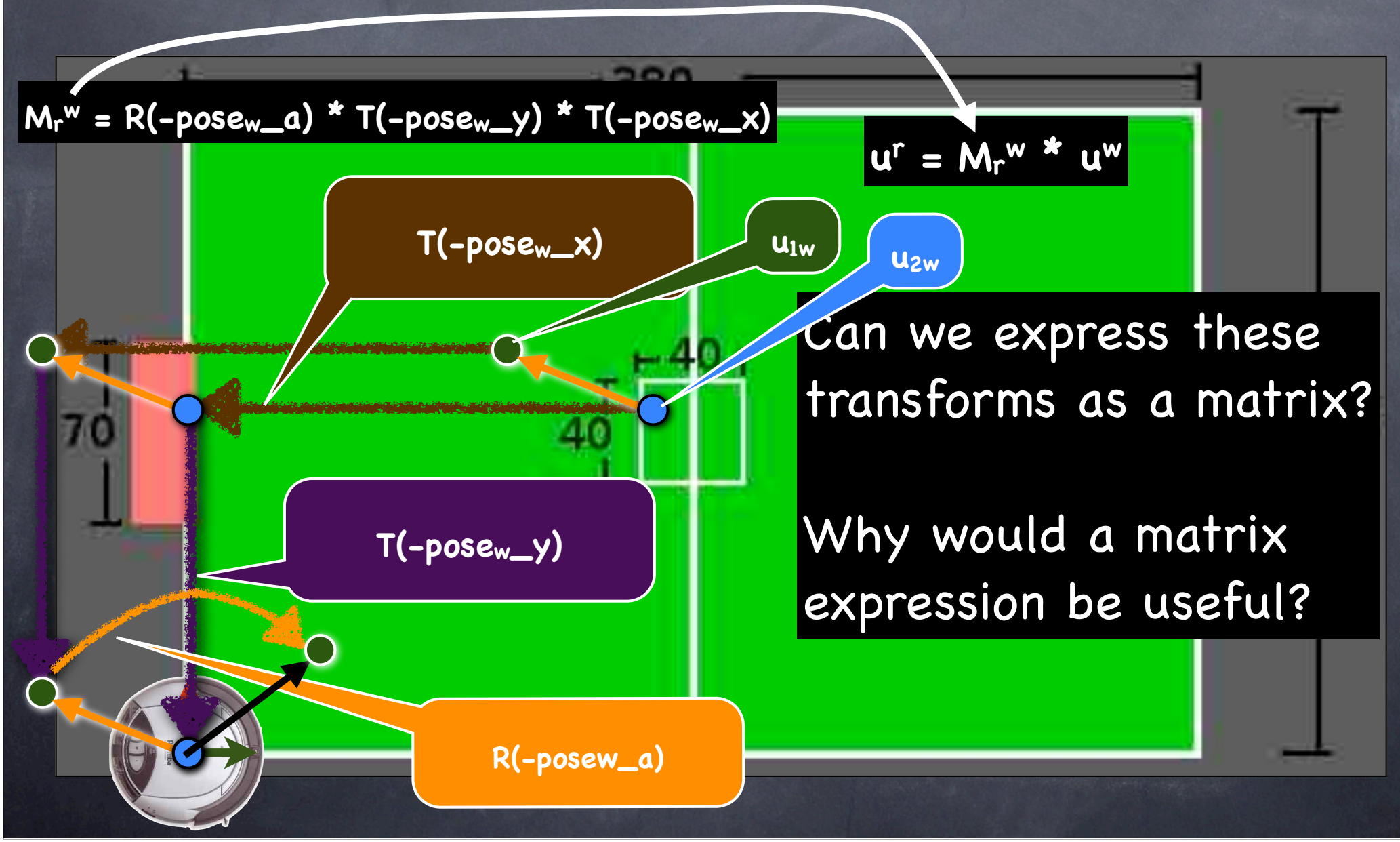
Pose space vector results from undoing world space translations and rotations on 2 points



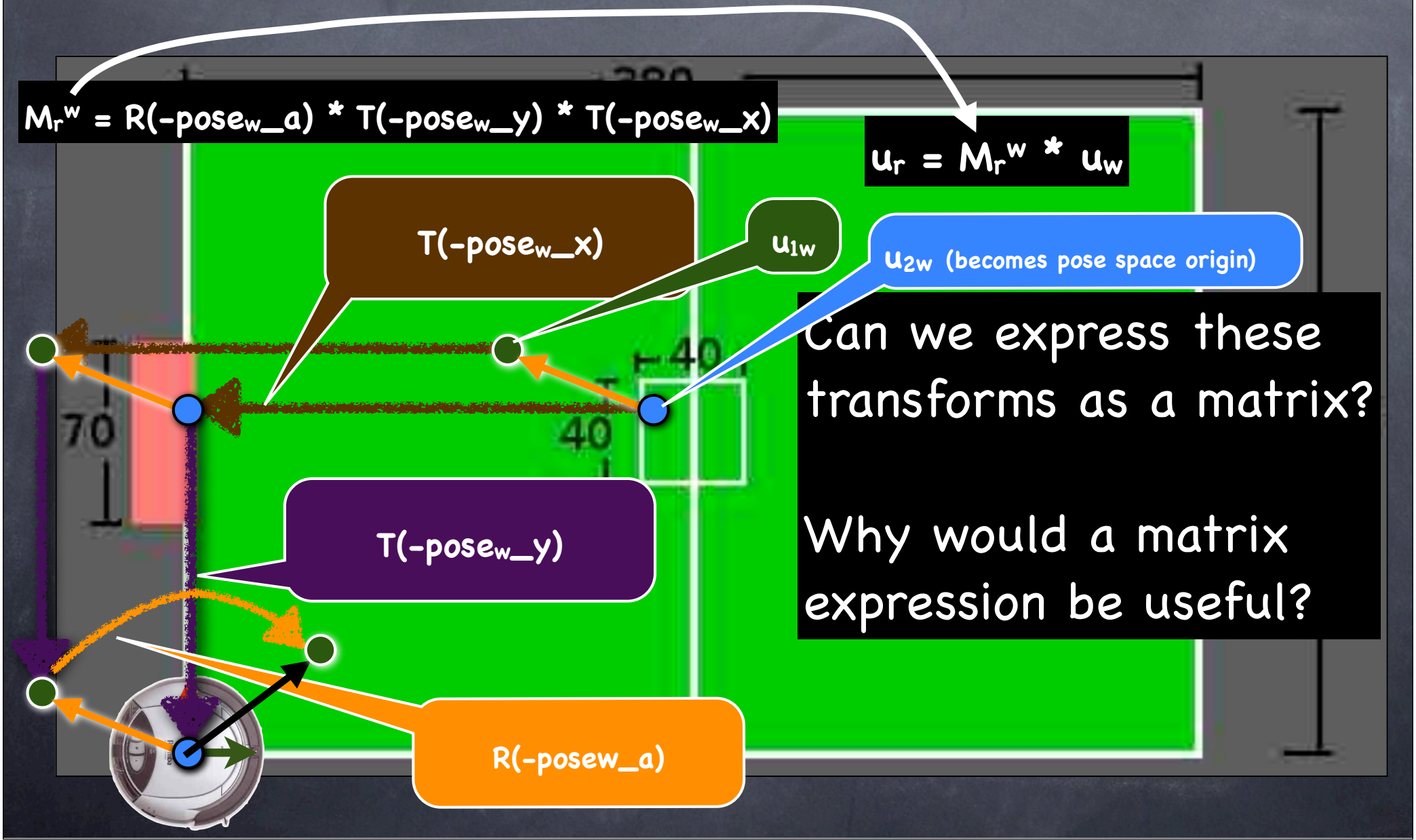
Pose space vector results from undoing world space translations and rotations on 2 points



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

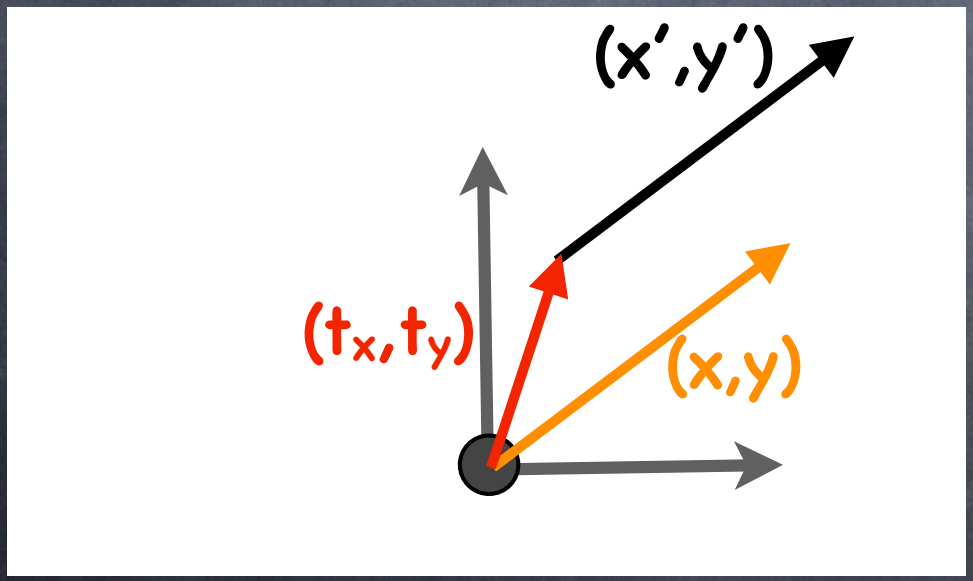


Are we all set to transform our vectors?



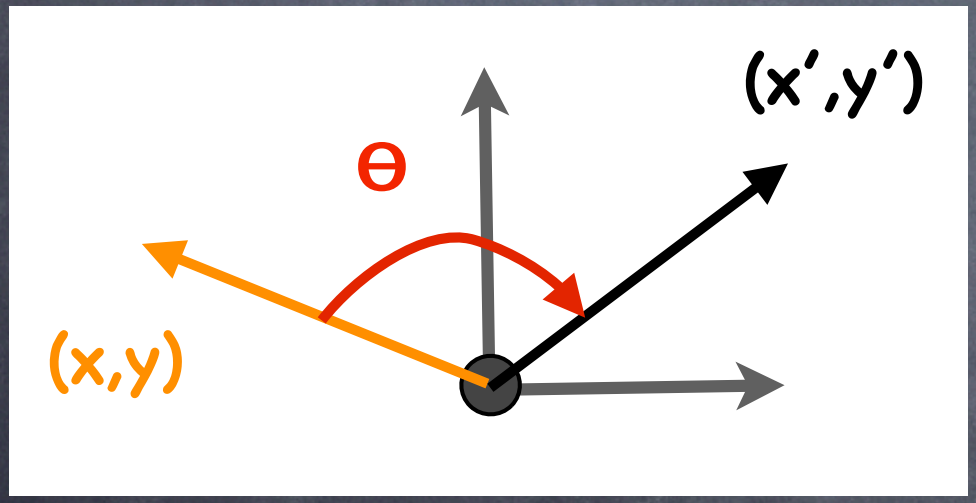
2D Translation

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

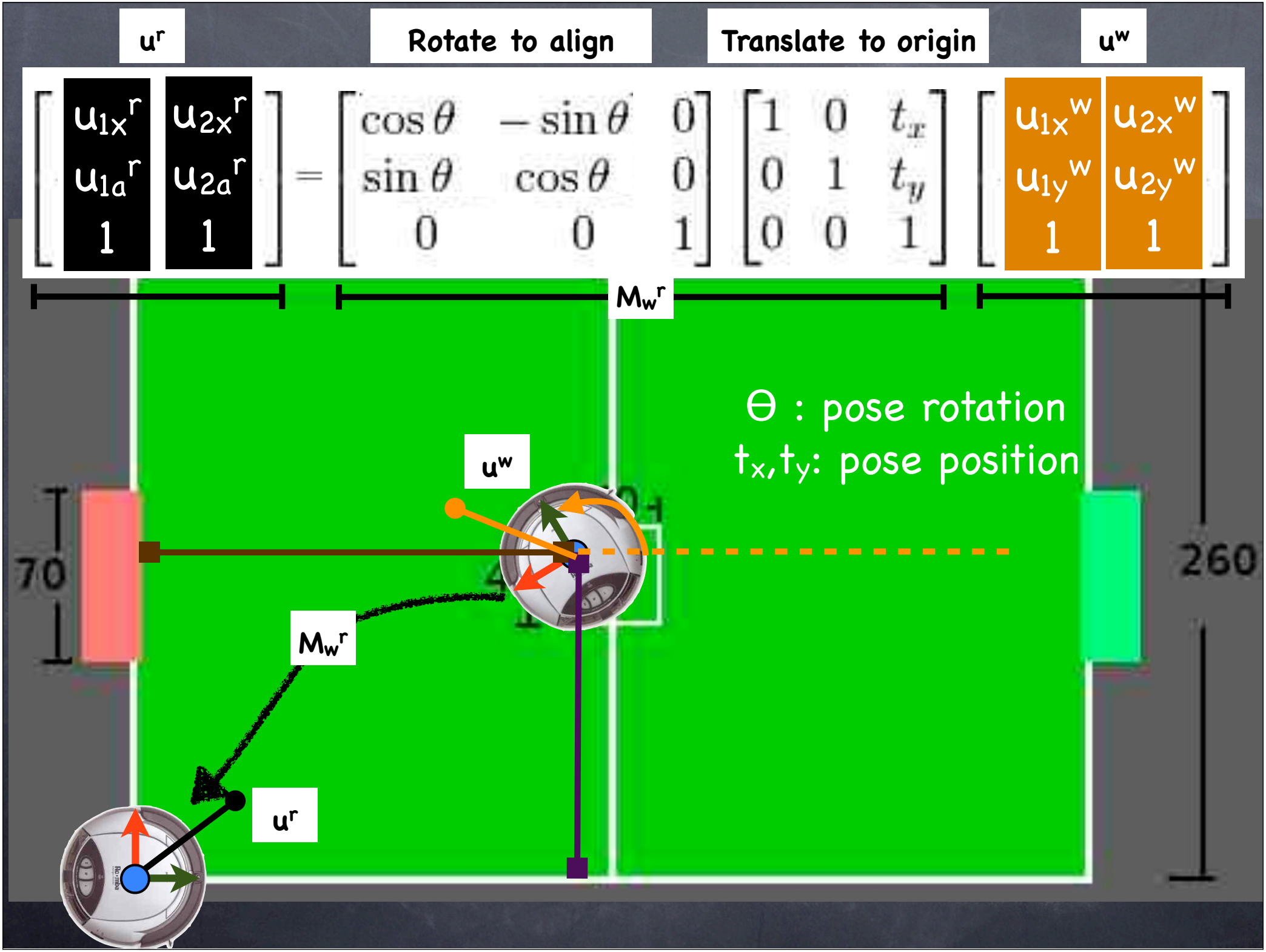


- Requires "homogeneous coordinates"
- 3D vector with position concatenated with a 1
- Matrix parameterized by horizontal and vertical displacement

Homogeneous 2D rotation matrix

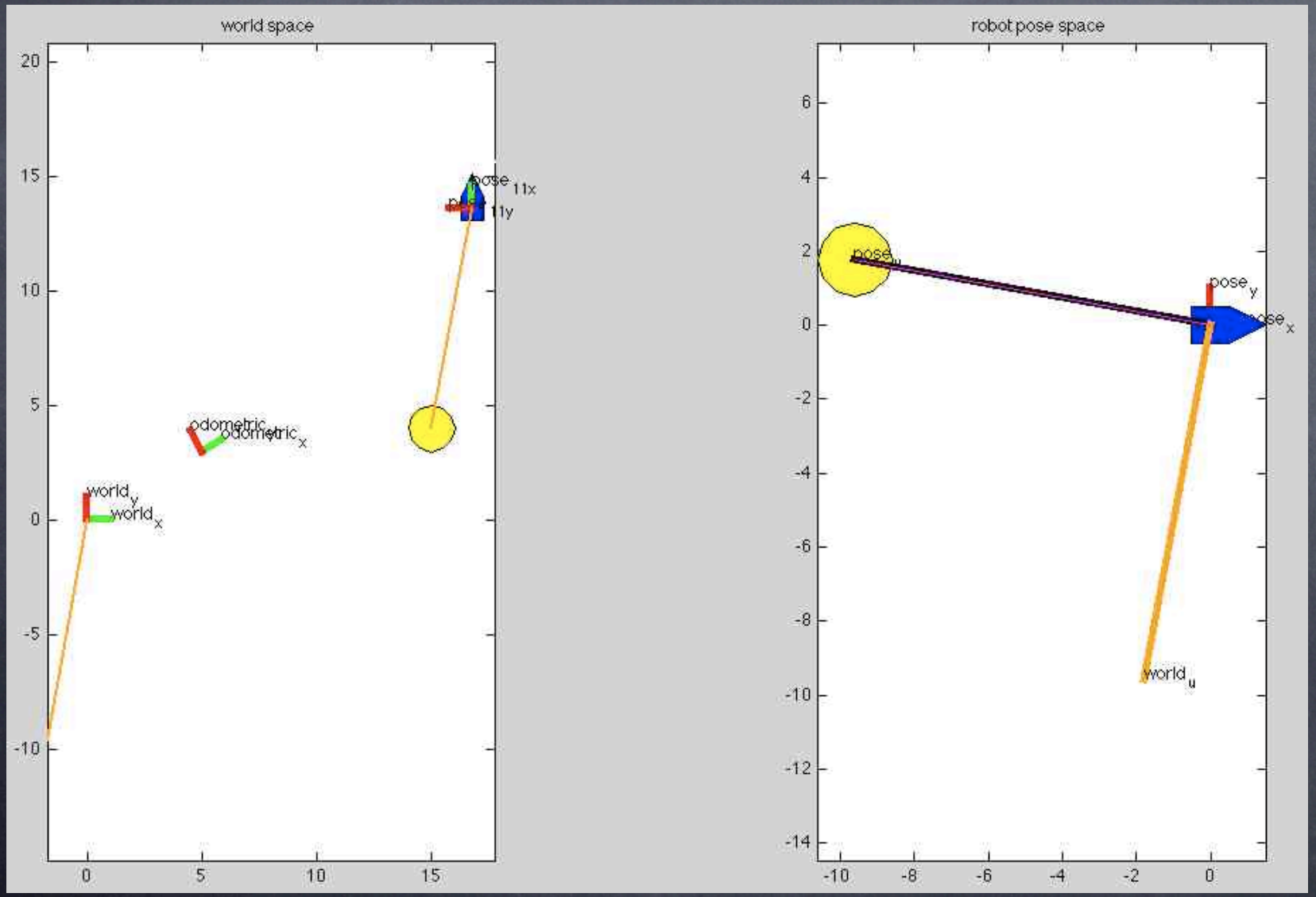


$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


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

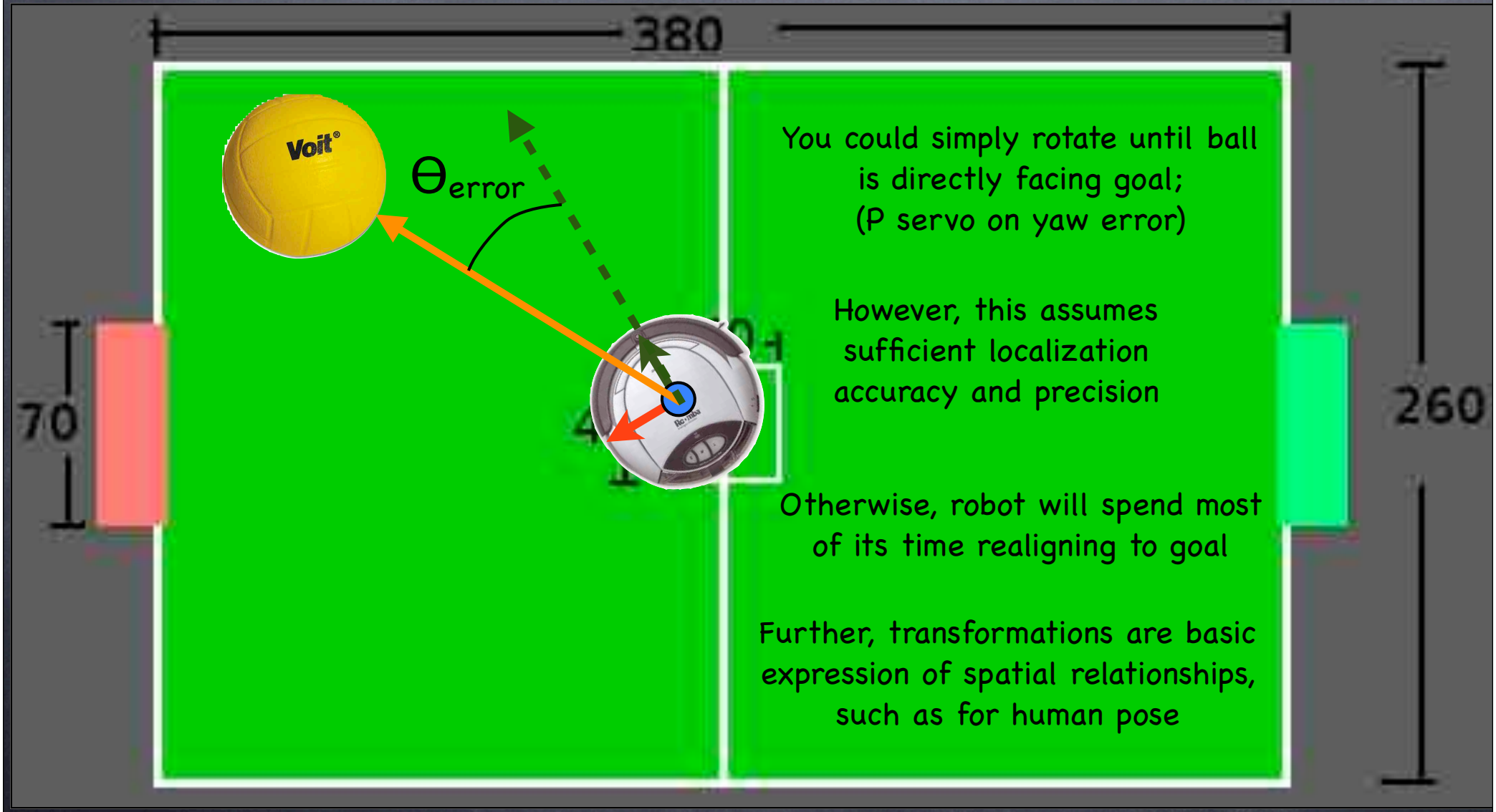
matlab example

- /course/cs148/pub/odometry_coordinate_transform.m



with odometry

Why bother with transforms?



You could simply rotate until ball is directly facing goal; (P servo on yaw error)

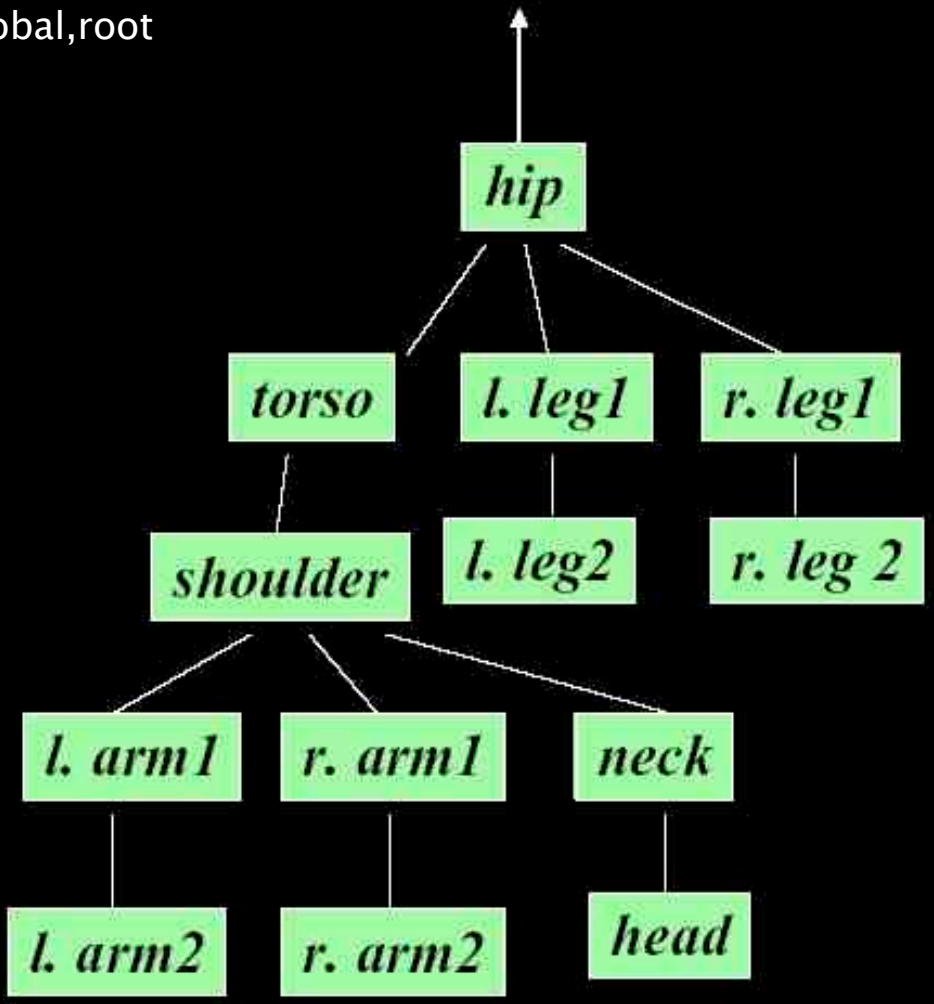
However, this assumes sufficient localization accuracy and precision

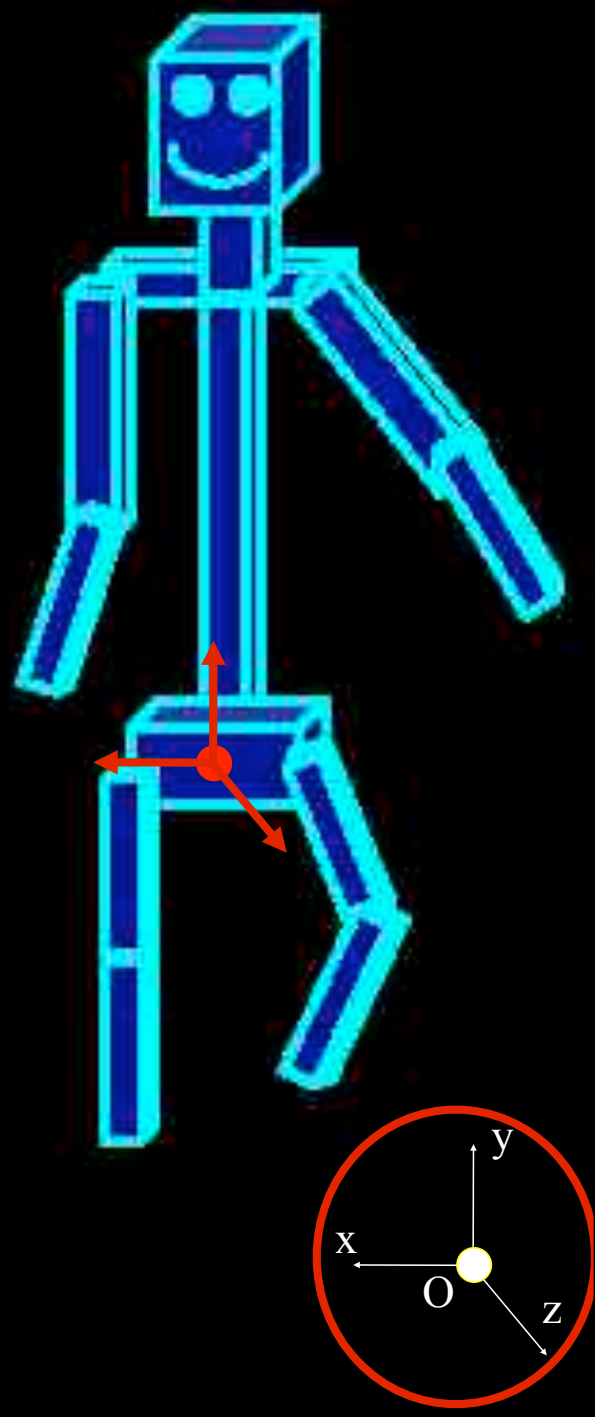
Otherwise, robot will spend most of its time realigning to goal

Further, transformations are basic expression of spatial relationships, such as for human pose

Human figure kinematics

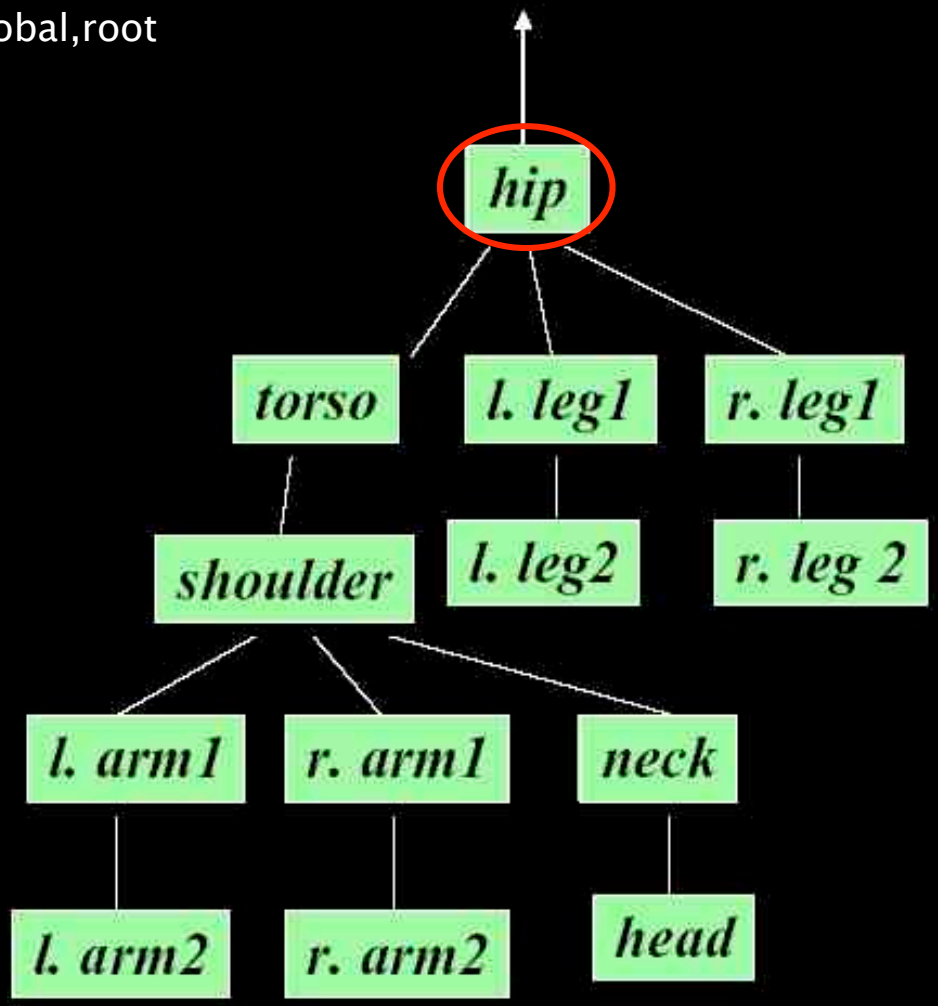
- Global coodinates $T_{global,root}$
  - absolute root

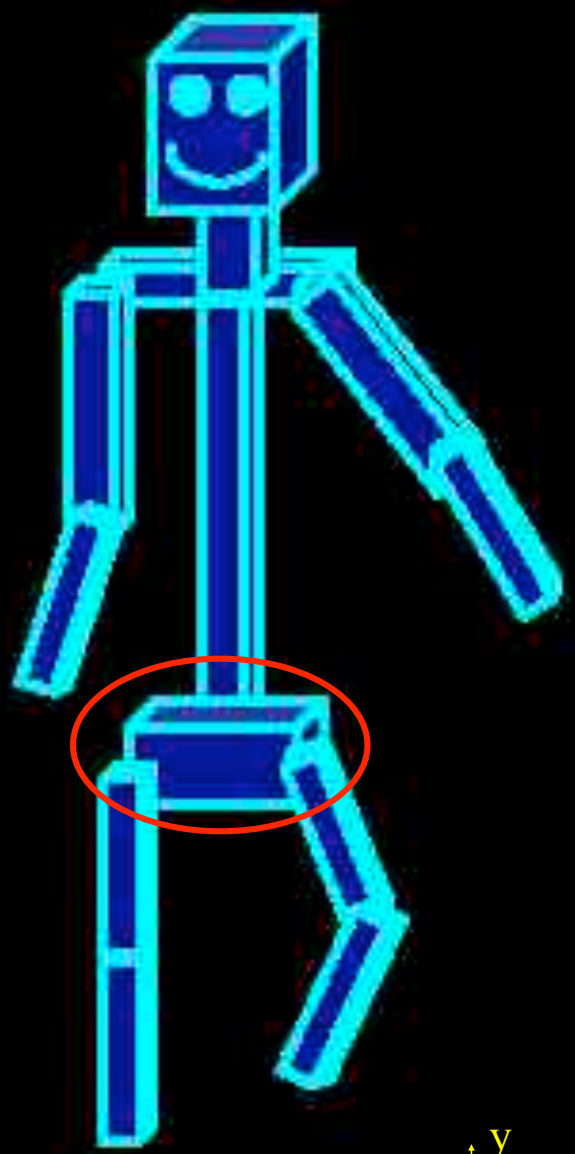




Human figure kinematics

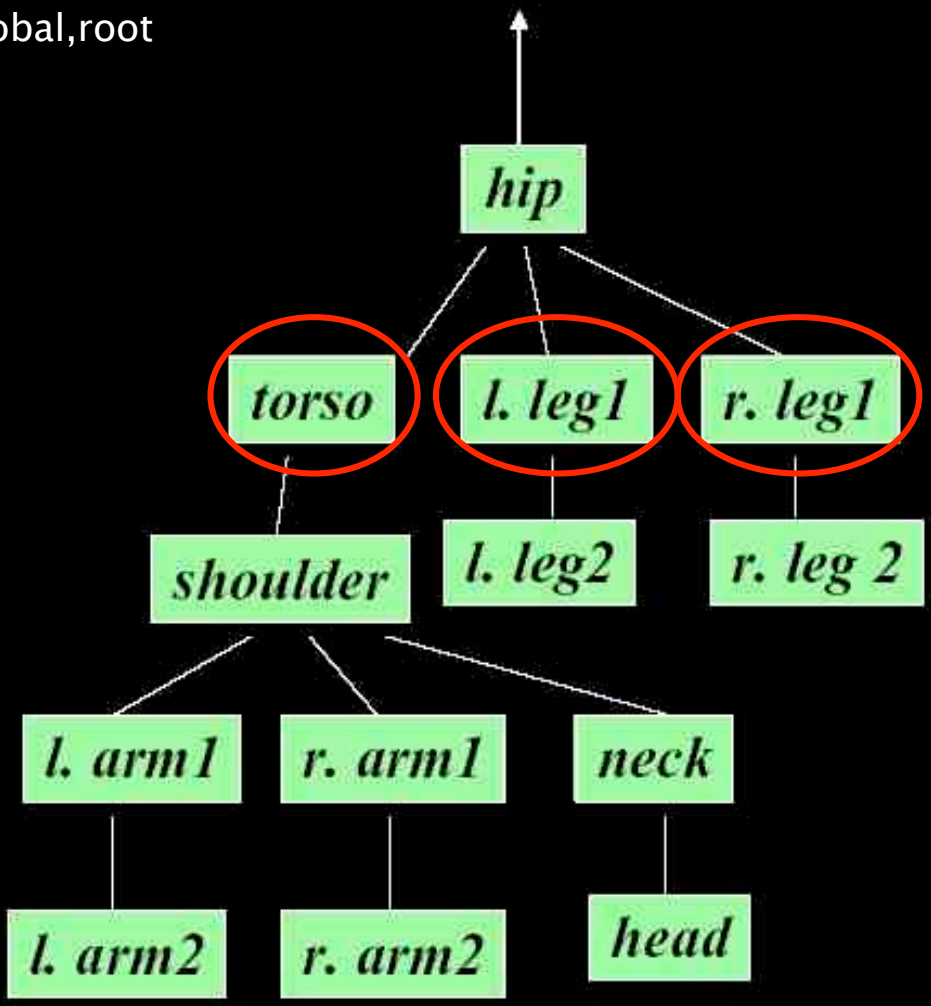
- Global coodinates $T_{global,root}$
  - absolute root
- Body root

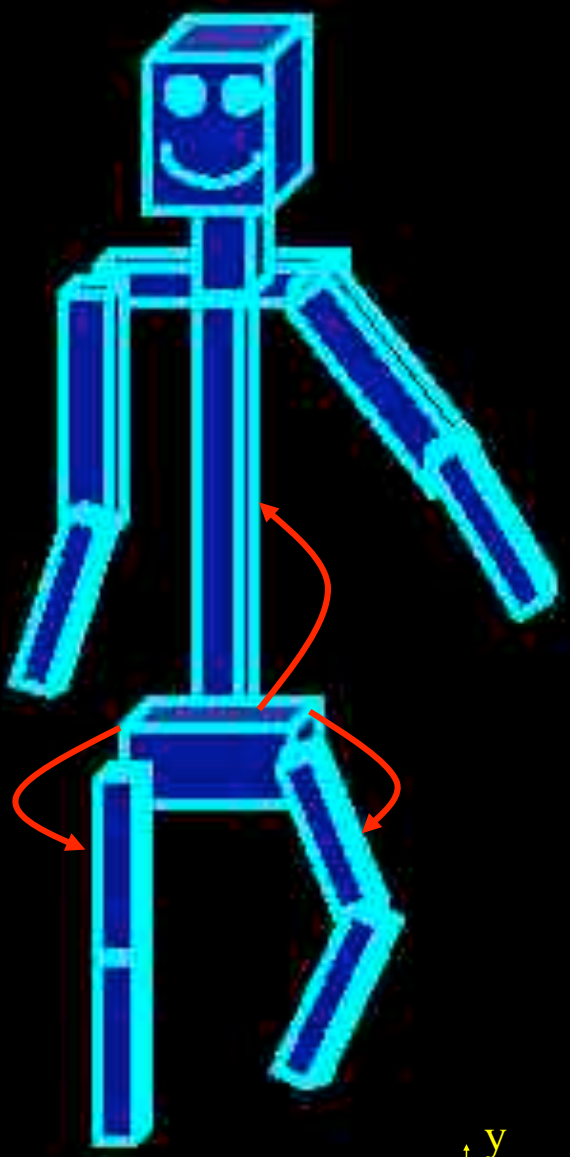


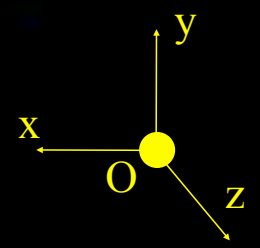


Human figure kinematics

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

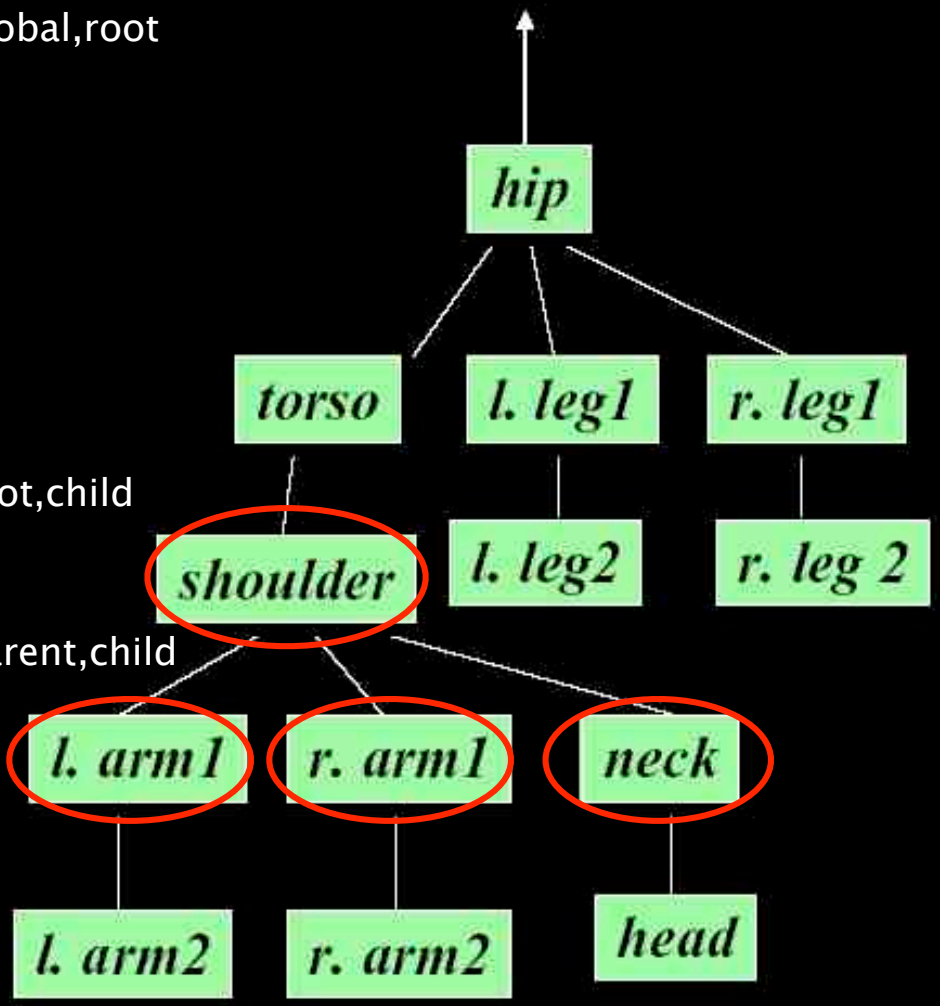


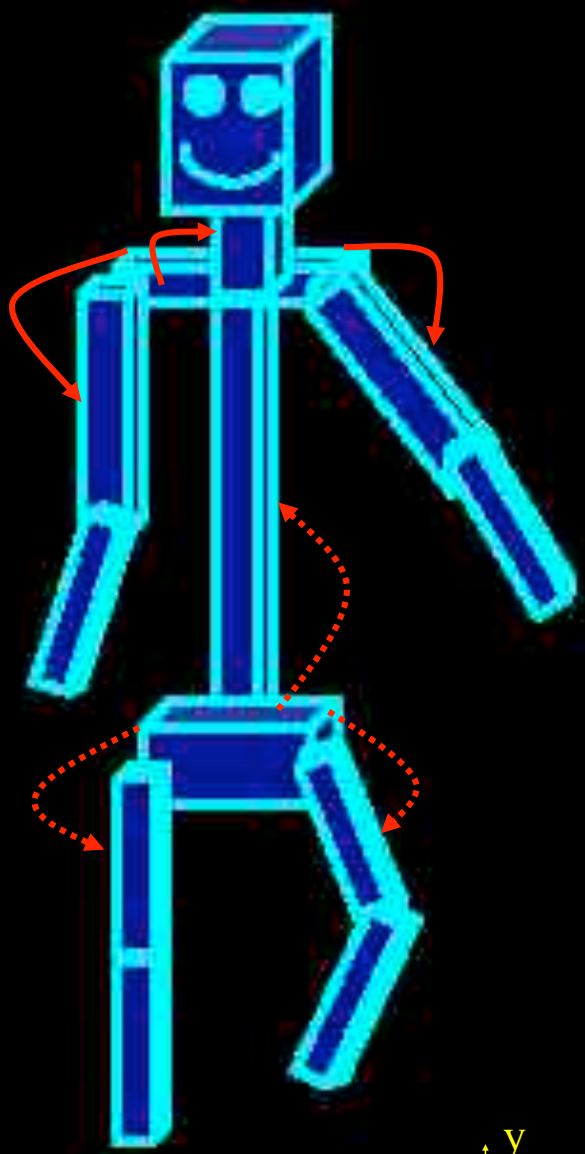




Human figure kinematics

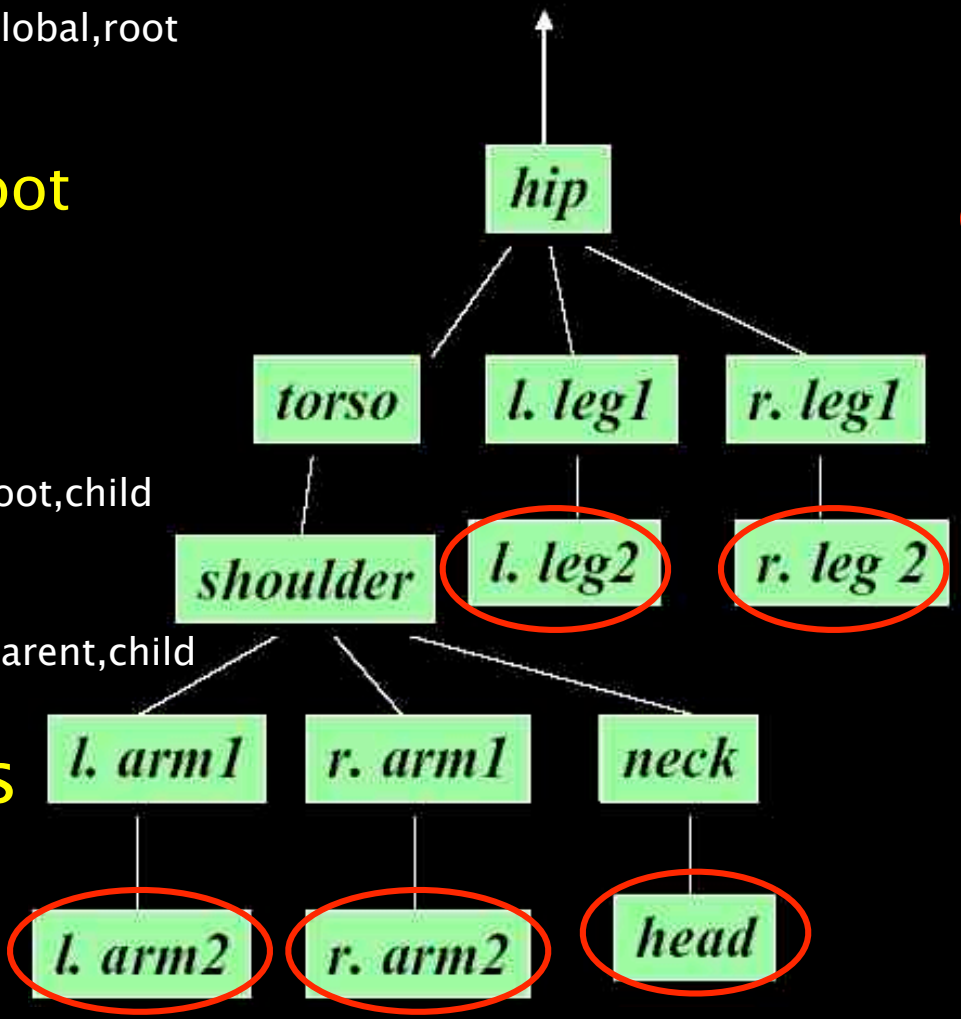
- Global coodinates $T_{global,root}$
  - absolute root
- Body root
- 1st level children $T_{root,child}$
- Nth level children $T_{parent,child}$

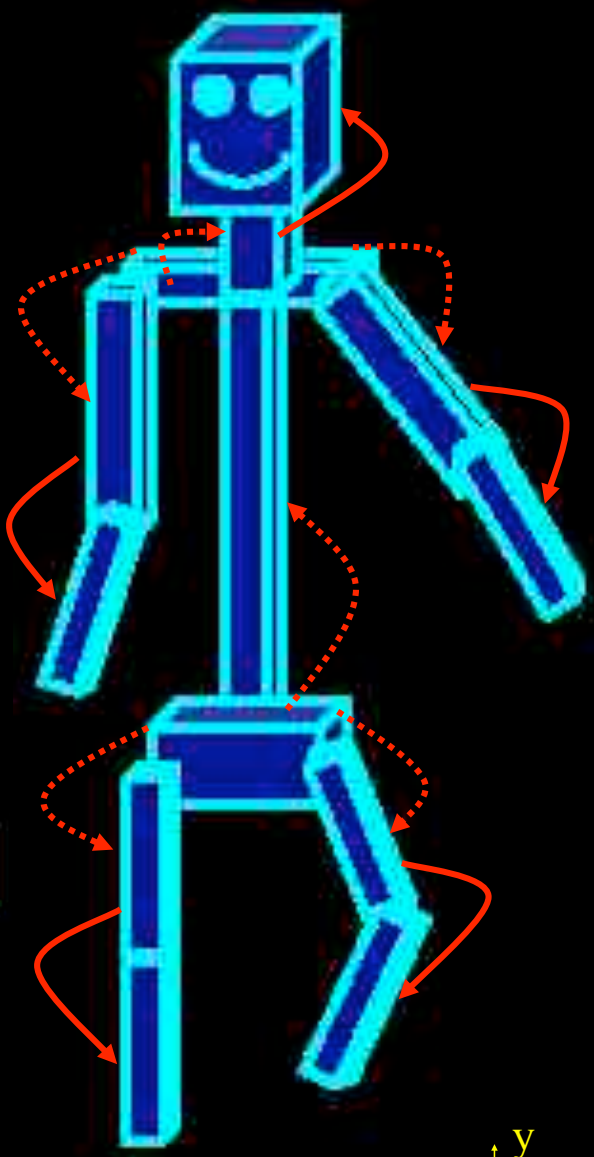


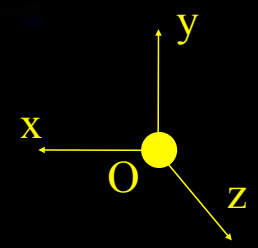


Human figure kinematics

- 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