

# Kinematics

15-494 Cognitive Robotics  
David S. Touretzky &  
Ethan Tira-Thompson

Carnegie Mellon  
Spring 2007

# Outline

Kinematics is the study of how things move.

- Homogeneous coordinates
- Kinematic chains
  - Description of the AIBO as a collection of chains
- Reference frames
- Kinematics and PostureEngine classes
- Forward kinematics: calculating limb positions from joint angles. (Straightforward math.)
- Inverse kinematics: calculating joint angles to achieve desired limb positions. (Hard.)

# Homogeneous Coordinates

- Represent a point in N-space by an (N+1)-dimensional vector. Extra component is an inverse scale factor.
  - In “normal” form, last component is 1.

$$\vec{v} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Points at infinite distance: last component is 0.
- Allows us to perform a variety of transformations using matrix multiplication:

Rotation, Translation, Scaling
- Tekkotsu uses 3D coordinates (so 4-dimensional vectors) for everything.

# NEWMAT

- Tekkotsu uses the NEWMAT package for vector and matrix arithmetic.

```
NEWMAT::ColumnVector v(4), w(4);  
v = Kinematics::pack(5.75, 30.0, 115);  
w << 17 << -4.2f << 100 << 1;
```

```
NEWMAT::Matrix T(4,4);  
T = v * w.t();
```

# Transformation Matrices

- Let  $\theta$  be rotation angle in the x-y plane.  
Let  $dx$ ,  $dy$ ,  $dz$  be translation amounts.  
Let  $1/s$  be a scale factor.

$$T = \begin{bmatrix} \cos \theta & \sin \theta & 0 & dx \\ -\sin \theta & \cos \theta & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & s \end{bmatrix}$$

$$T \vec{V} = \begin{bmatrix} x \cos \theta + y \sin \theta + dx \\ -x \sin \theta + y \cos \theta + dy \\ z + dz \\ \textcolor{red}{s} \end{bmatrix} = \begin{bmatrix} (x \cos \theta + y \sin \theta + dx)/s \\ (-x \sin \theta + y \cos \theta + dy)/s \\ (z + dz)/s \\ \textcolor{red}{1} \end{bmatrix}$$

# Transformations Are Composable

- To rotate about point  $p$ , translate  $p$  to the origin, rotate, then translate back.

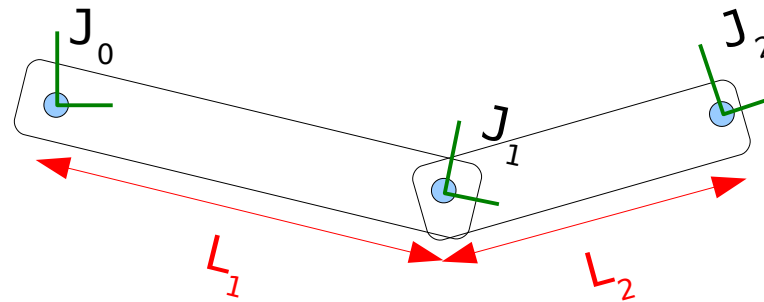
$$\textit{Translate}(p) = \begin{bmatrix} 1 & 0 & 0 & p.x \\ 0 & 1 & 0 & p.y \\ 0 & 0 & 1 & p.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\textit{Rotate}(\theta) = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\textit{RotateAbout}(p, \theta) = \textit{Translate}(p) \cdot \textit{Rotate}(\theta) \cdot \textit{Translate}(-p)$$

# Kinematic Chains

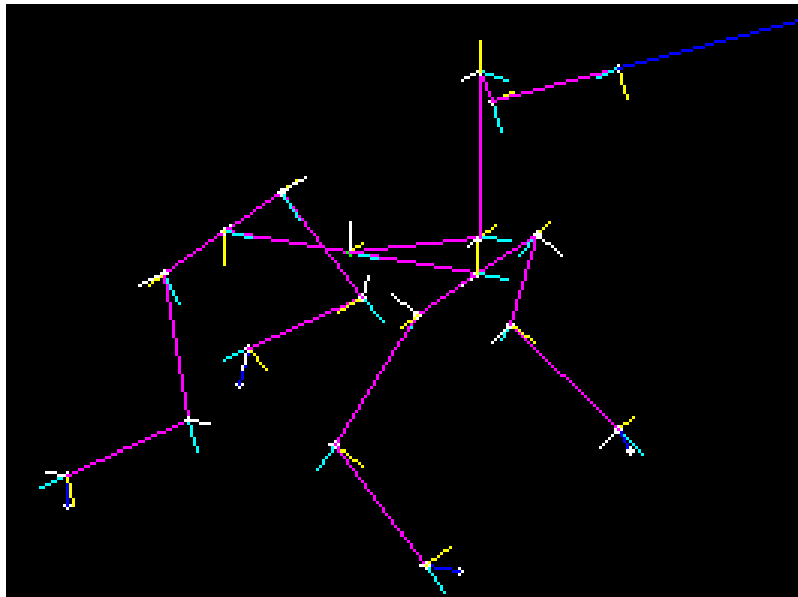
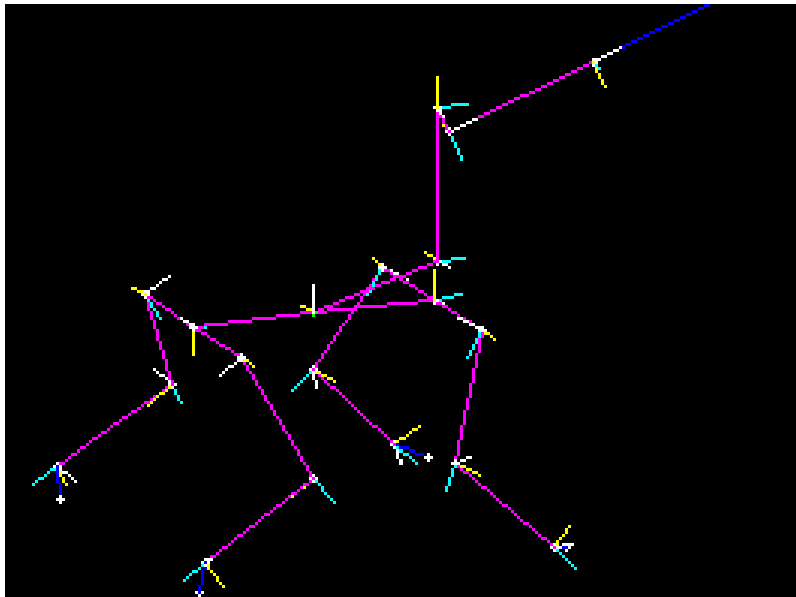
- Sequence of joints separated by links.



- We can use transformation matrices to calculate the position of the tip of the chain (joint  $J_2$ ) from the joint angles  $\theta_0$ ,  $\theta_1$  and the link lengths  $L_1$ ,  $L_2$ .
- Each joint has a rotation transform; each link has a translation transform.

# AIBO Kinematic Chains

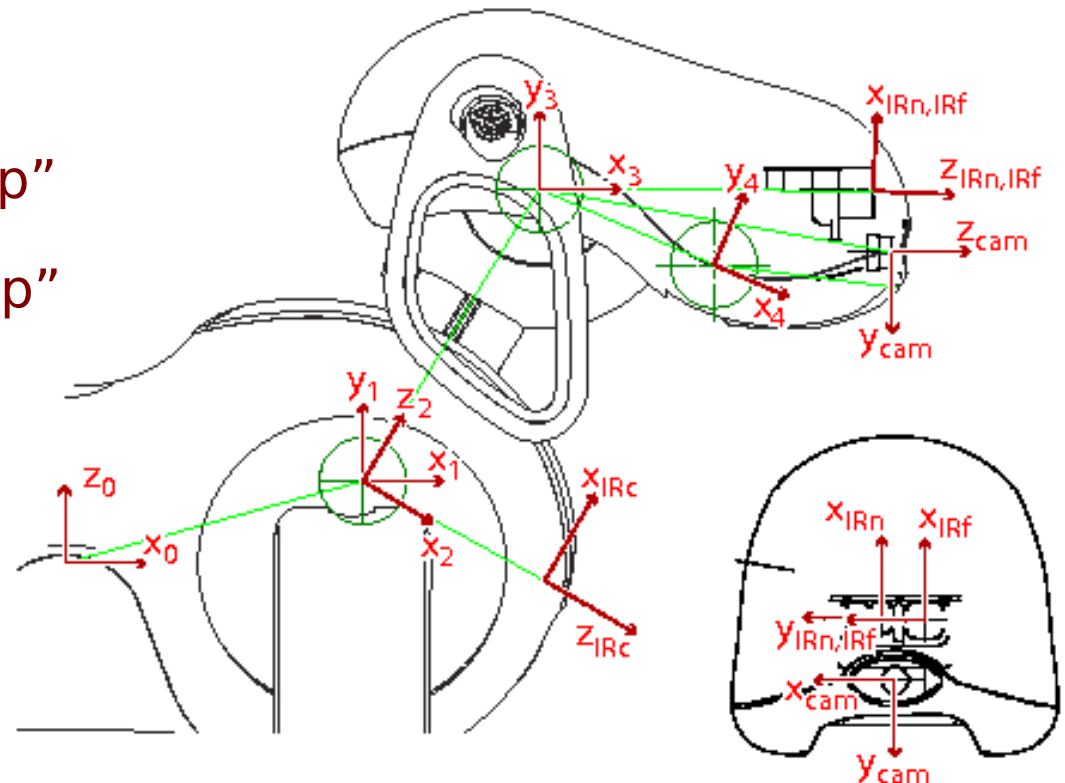
- The AIBO has 9 kinematic chains:
  - 4 for the legs
  - 1 for the head (ending in the camera), 1 for the mouth
  - 3 for the IR range sensors
- All chains begin at the center of the body (base frame).





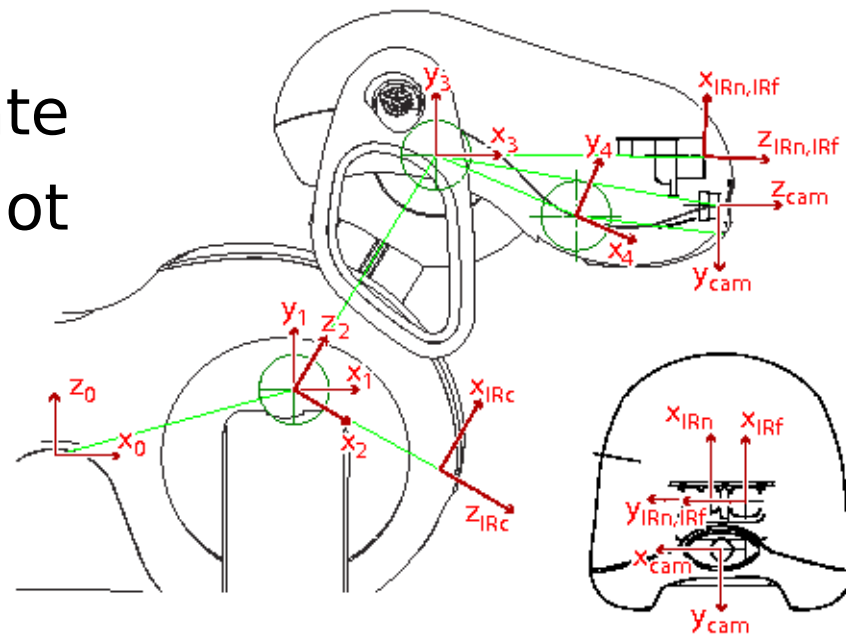
# Reference Frames

- Every joint and every link has an associated reference frame.
- Denavit-Hartenberg conventions: all joints move about their reference frame's z-axis.
- The head chain:
  - Base frame 0  $z_0 = \text{"up"}$
  - Tilt joint 1  $y_1 = \text{"up"}$
  - Pan joint 2
  - Nod joint 3
  - Camera 4

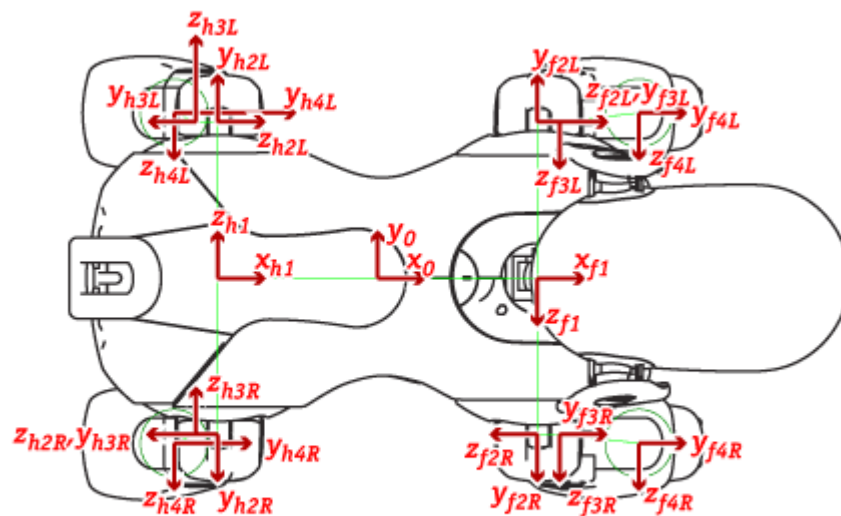


# Joint vs. Link Reference Frames

- The joint reference frame does not rotate with the joint. The link reference frame does.
- The  $x_1, y_1, z_1$  joint axes remain fixed with respect to the base frame when the head tilts up or down.
- The  $x_2, z_2$  joint axes rotate with the tilt angle (but not the pan angle.)



# Leg Reference Frames



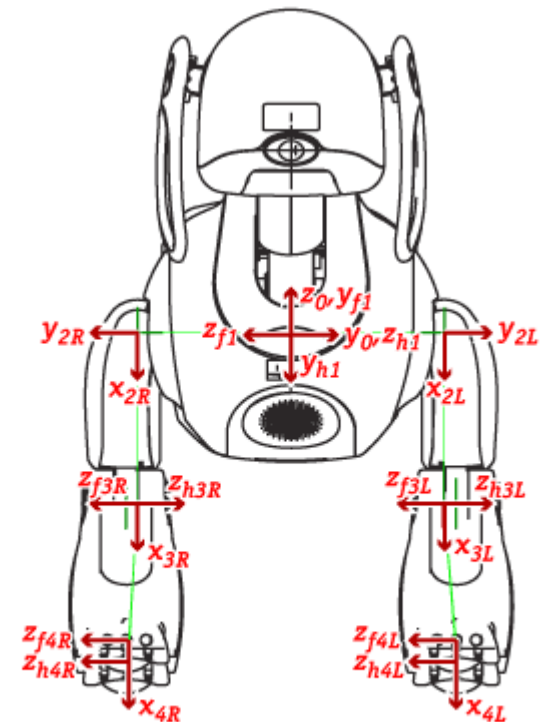
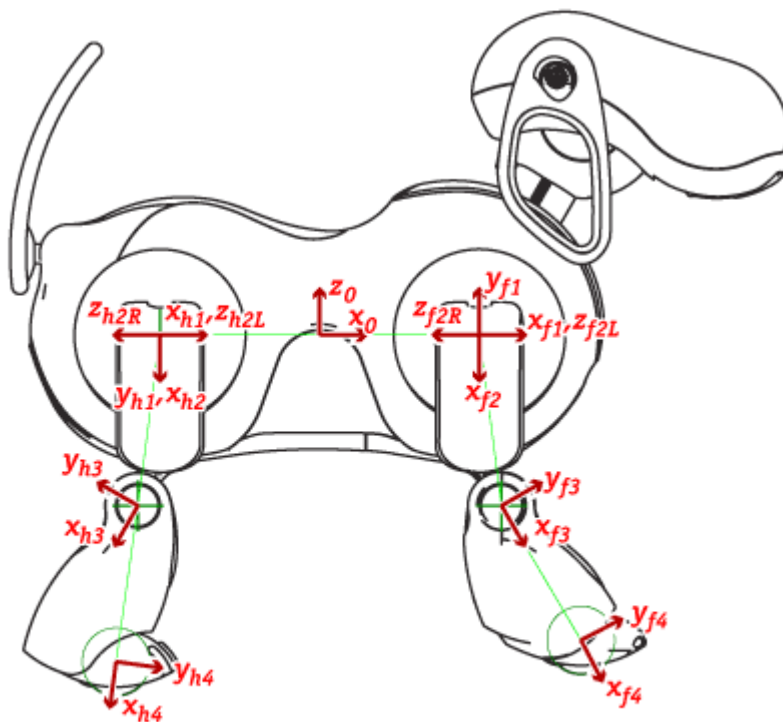
ERS-7 Legs

	$\Delta x$	$\Delta y$	$\Delta z$
1. - shoulder	65	0	0
2. - elevator	0	0	62.5
3. - knee	69.5	0	9
f4. - ball	69.987	-4.993	4.7
h4. - ball	67.681	-18.503	4.7

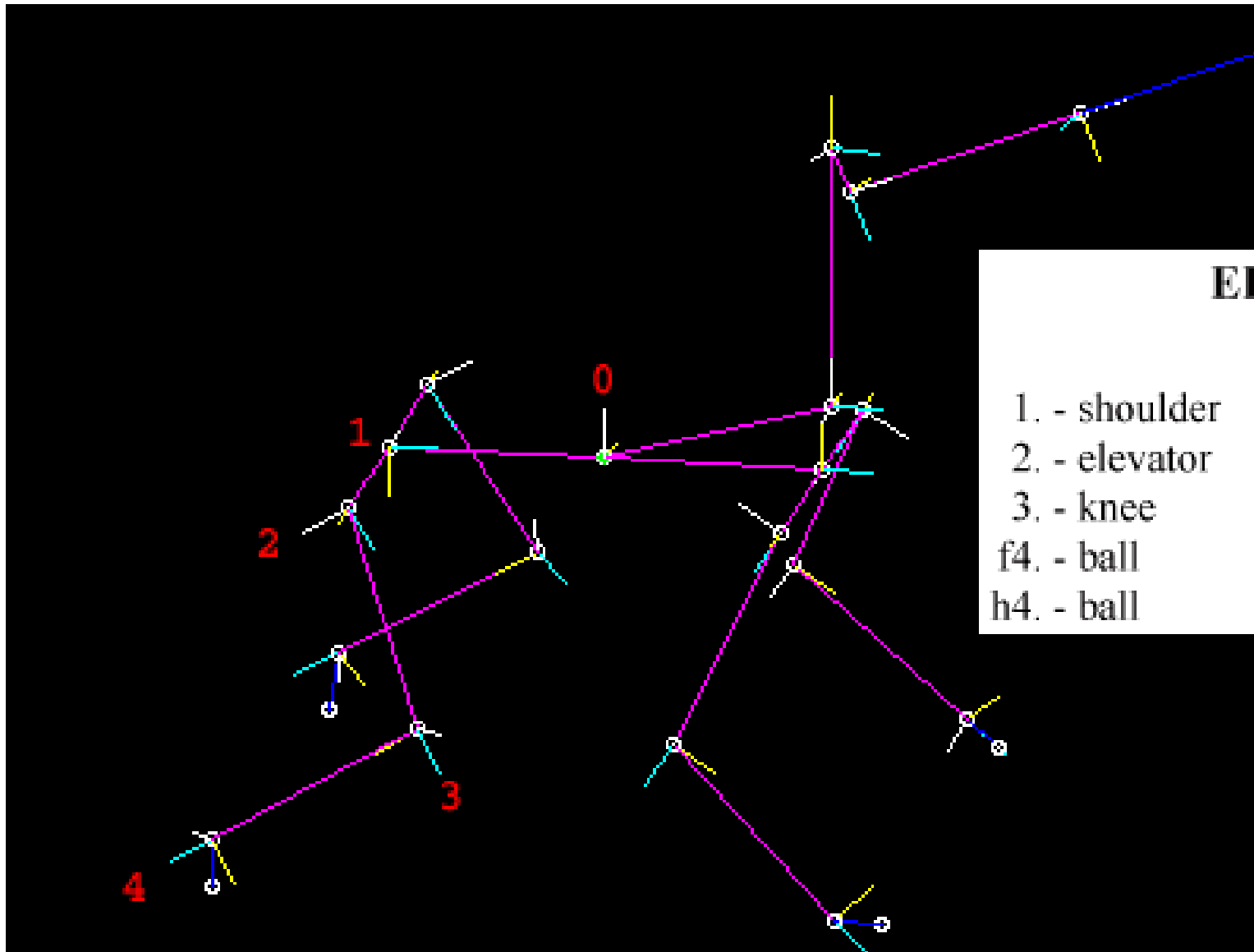
Diameter of ball of foot is 23.433mm

Each link offset is relative to previous link

The shins shown in this diagram appear to be slightly distorted compared to a real robot. Corresponding measurements have been taken from actual models.



# Leg Reference Frames



**ERS-7 Legs**

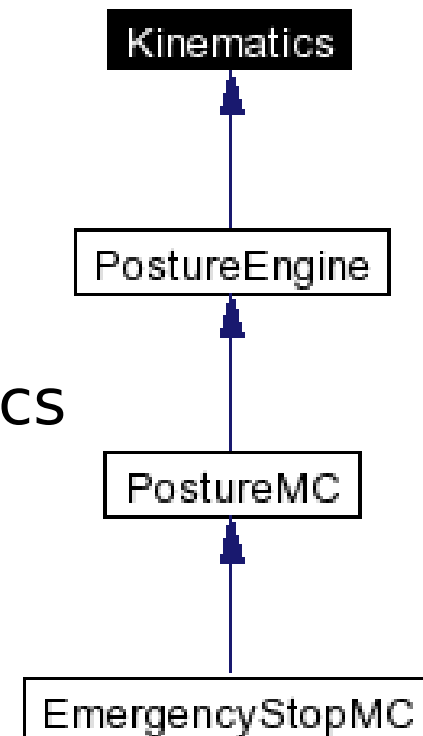
	$\Delta x$	$\Delta y$	$\Delta z$
1. - shoulder	65	0	0
2. - elevator	0	0	62.5
3. - knee	69.5	0	9
f4. - ball	69.987	-4.993	4.7
h4. - ball	67.681	-18.503	4.7

# Reference Frame Naming Conventions

- Use a similar offset-based indexing scheme as for joint names in motion commands and world state vectors:
  - BaseFrameOffset
  - HeadOffset + TiltOffset
  - CameraOffset
  - LFrLegOffset + ElevatorOffset
- Note: the distinction between joint and link reference frames is made in the function name, not the reference frame name:
  - jointToBase(HeadOffset+TiltOffset)
  - linkToBase(HeadOffset+TiltOffset)

# Kinematics Class

- Tekkotsu uses the ROBOOP package for kinematics calculations.
- The Kinematics class provides access to ROBOOP functionality for forward kinematics.
- Global variable **kine** holds a special Kinematics instance:
  - Joint values reference WorldState
- PostureEngine is a child of Kinematics so it can do kinematics calculations too. It adds inverse kinematics.
  - Joint angle results are stored in the PostureEngine instance.



# Converting Between Reference Frames

- Most common conversion is between the base frame (body coordinates) and a limb frame.

baseToJoint(), baseToLink(), jointToBase(), linkToBase()

NEWMAT::ReturnMatrix jointToBase(unsigned int joint)

- General conversion functions:

jointToJoint(), jointToLink(), linkToJoint(), linkToLink()

NEWMAT::ReturnMatrix jointToJoint(unsigned int ij,  
unsigned int oj)

# Reference Frame Conversion 1

- Transform Base to Base:

```
cout.precision(3);  
cout << setw(7) << kine->jointToBase(BaseFrameOffset);  
cout << endl;
```

- Result:

1.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000
0.000	0.000	1.000	0.000
0.000	0.000	0.000	1.000



# Reference Frame Conversion 2

- Translate base frame to head tilt frame:

```
const float headtilt = state->outputs[HeadOffset+TiltOffset];
cout << "Head tilt is " << headtilt * 360/(2*M_PI)
    << " degrees." << endl;

NEWMAT::Matrix TtiltJ(kine->jointToBase(HeadOffset+TiltOffset));
NEWMAT::Matrix TtiltL(kine->linkToBase (HeadOffset+TiltOffset));

cout << "tilt jointToBase= " << endl << setw(7) << TtiltJ << endl;
cout << "tilt linkToBase= " << endl << setw(7) << TtiltL << endl;
```

# At ~Zero Degree Tilt Angle

Head tilt is 1.25 degrees.

tilt jointToBase=

1.000	0.000	0.000	67.500
0.000	0.000	-1.000	0.000
0.000	1.000	0.000	19.500
0.000	0.000	0.000	1.000

tilt linkToBase=

1.000	-0.022	0.000	67.500
0.000	0.000	-1.000	0.000
0.022	1.000	0.000	19.500
0.000	0.000	0.000	1.000

## ERS-7 Head

	$\Delta x$	$\Delta y$	$\Delta z$
1. - tilt <sub>0</sub>	67.5	0	19.5
2. - pan <sub>1</sub>	0	0	0
3. - nod <sub>2</sub>	0	0	80
4. - jaw <sub>3</sub>	40	-17.5	0
cam. - camera <sub>3</sub>	81.06	-14.6	0
IRn. - NearIR <sub>3</sub>	76.9	1.917	2.795
IRf. - FarIR <sub>3</sub>	76.9	1.052	-8.047
IRc. - ChestIR <sub>0</sub>	109.136	-3.384	0

$$x_3 \angle x_4 = -23.6294^\circ$$

# At $\sim -30$ Degree Tilt Angle

Head tilt is  $-29.5$  degrees.

tilt jointToBase=

1.000	0.000	0.000	67.500
0.000	0.000	-1.000	0.000
0.000	1.000	0.000	19.500
0.000	0.000	0.000	1.000

tilt linkToBase=

0.871	0.492	0.000	67.500
0.000	0.000	-1.000	0.000
-0.492	0.871	0.000	19.500
0.000	0.000	0.000	1.000

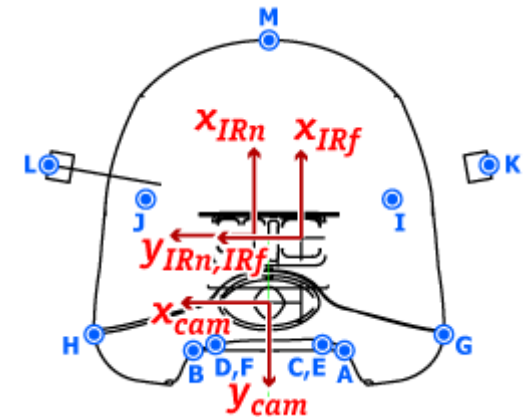
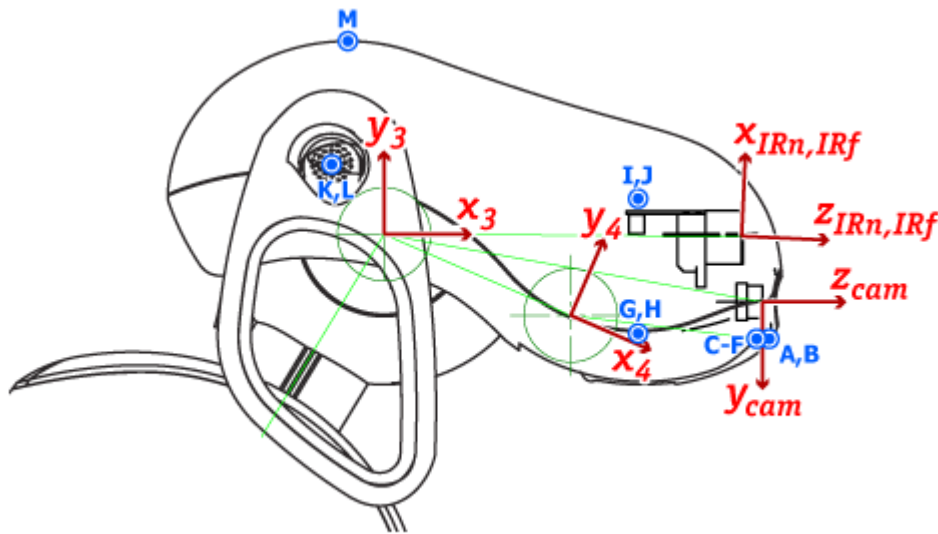
$\cos(-30^\circ) = 0.866$

$\sin(-30^\circ) = 0.500$

- The tilt joint reference frame doesn't rotate with tilt.
- The tilt link reference frame does rotate.

# Interest Points

- Interest points on the head, legs, and body are predefined for use in kinematics calculations.



## Interest Points:

- A - LowerLeftLowerLip<sub>4</sub>
- B - LowerRightLowerLip<sub>4</sub>
- C - UpperLeftLowerLip<sub>4</sub>
- D - UpperRightLowerLip<sub>4</sub>
- E - LowerLeftUpperLip<sub>3</sub>
- F - LowerRightUpperLip<sub>3</sub>
- G - LowerLeftSnout<sub>3</sub>
- H - LowerRightSnout<sub>3</sub>
- I - UpperLeftSnout<sub>3</sub>
- J - UpperRightSnout<sub>3</sub>
- K - LeftMicrophone<sub>3</sub>
- L - RightMicrophone<sub>3</sub>
- M - HeadButton<sub>3</sub>

# Leg Interest Points

## Interest Points:

- A - Toe{L,R}{Fr,Bk}Paw<sub>4</sub>
- B - Lower{Inner,Outer}Front{L,R}{Fr,Bk}Shin<sub>3</sub>
- C - Lower{Inner,Outer}Middle{L,R}{Fr,Bk}Shin<sub>3</sub>
- D - Lower{Inner,Outer}Back{L,R}{Fr,Bk}Shin<sub>3</sub>
- E - Middle{Inner,Outer}Middle{L,R}{Fr,Bk}Shin<sub>3</sub>
- F - Upper{Inner,Outer}Front{L,R}{Fr,Bk}Shin<sub>3</sub>
- G - Upper{Inner,Outer}Back{L,R}{Fr,Bk}Shin<sub>3</sub>
- H - Lower{Inner,Outer}Front{L,R}{Fr,Bk}Thigh<sub>2</sub>
- I - Lower{Inner,Outer}Back{L,R}{Fr,Bk}Thigh<sub>2</sub>
- J - Upper{Inner,Outer}Front{L,R}{Fr,Bk}Thigh<sub>2</sub>
- K - Upper{Inner,Outer}Back{L,R}{Fr,Bk}Thigh<sub>2</sub>
- L - Upper{L,R}Chest<sub>0</sub>
- M - Lower{L,R}Chest<sub>0</sub>
- N - {L,R}{Fr,Bk}Belly<sub>0</sub>
- O - Lower{L,R}Rump<sub>0</sub>
- P - Upper{L,R}Rump<sub>0</sub>

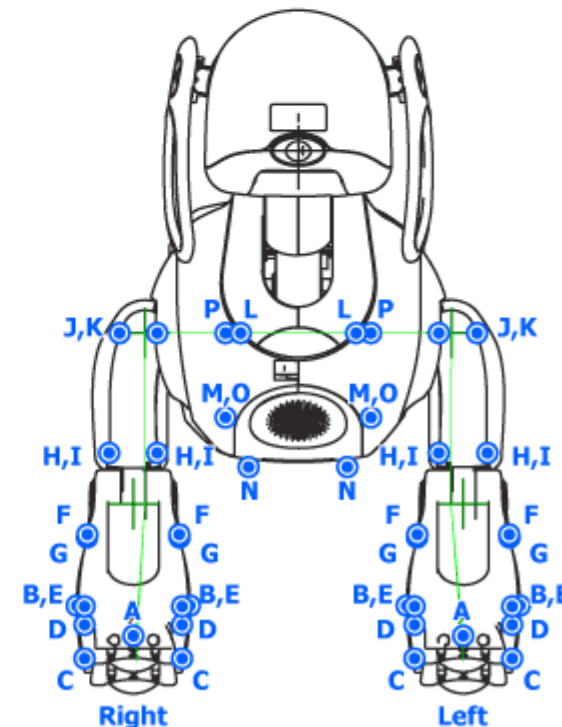
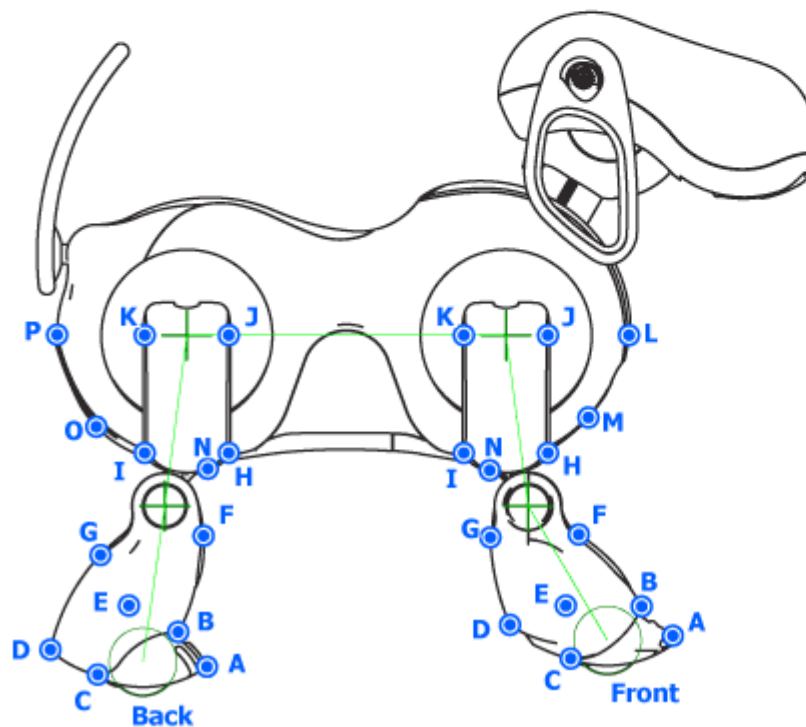
## ERS-7 Legs

	$\Delta x$	$\Delta y$	$\Delta z$
1. - shoulder	65	0	0
2. - elevator	0	0	62.5
3. - knee	69.5	0	9
f4. - ball	69.987	-4.993	4.7
h4. - ball	67.681	-18.503	4.7

Diameter of ball of foot is 23.433mm

Each link offset is relative to previous link

The shins shown in this diagram appear to be slightly distorted compared to a real robot.  
Corresponding measurements have been taken from actual models.



# Retrieving Interest Points

- Each interest point is attached to a link:  

```
void getInterestPoint(const std::string &name,  
                    unsigned int &link,  
                    NEWMAT::Matrix &coords)
```

  - Returns the link associated with the named interest point, and its coordinates in the link's reference frame.
- Interest points can be expressed in any reference frame:  

```
NEWMAT::ReturnMatrix  
getJointInterestPoint(unsigned int joint,  
                    const std::string &name)
```

# Forward Kinematics: Measure Distance Between Front Paws

```
virtual void processEvent(const EventBase&) {  
  
    NEWMAT::ColumnVector Pleftpaw =  
        kine->getJointInterestPoint(BaseFrameOffset,  
                                     "LowerInnerMiddleLFrShin");  
  
    NEWMAT::ColumnVector Prightpaw =  
        kine->getJointInterestPoint(BaseFrameOffset,  
                                     "LowerInnerMiddleRFrShin");  
  
    const float distance = (Pleftpaw-Prightpaw).NormFrobenius();  
  
    std::cout.setf(ios::fixed);  
    std::cout.precision(1);  
    std::cout << "Inter-paw distance is "  
                << setw(5) << distance << " mm." << std::endl;  
  
}
```

# Inverse Kinematics: lookAtPoint

- Inverse kinematics finds the joint angles to put an effector at a particular point in space.
- Hard problem:
  - solution space can be discontinuous
  - can be highly nonlinear
  - multiple solutions may be possible
  - maybe no solution (so find closest approximation)
- Example: `lookAtPoint(x,y,z,distance)`
  - point described in base frame coordinates
  - calculates head joint angles



# Stare At Paw Behavior 1

```
class DstBehavior : public BehaviorBase {
private:
    MotionManager::MC_ID relax_id, headpointer_id;
    static const float head_to_paw_distance = 30.0; // millimeters

public:
    DstBehavior() : BehaviorBase("DstBehavior"),
                   relax_id(MotionManager::invalid_MC_ID),
                   headpointer_id(MotionManager::invalid_MC_ID){}

    virtual void DoStart() {
        BehaviorBase::DoStart();

        relax_id =
            motman->addPersistentMotion(SharedObject<PIDMC>
                                         (LFrLegOffset, LFrLegOffset+JointsPerLeg, 0.0));

        headpointer_id =
            motman->addPersistentMotion(SharedObject<HeadPointerMC>());

        erouter->addListener(this, EventBase::sensorEGID);
    }
}
```

# Stare At Paw Behavior 2

```
virtual void processEvent(const EventBase&) {  
  
    NEWMAT::ColumnVector Ptarget =  
        kine->getJointInterestPoint(BaseFrameOffset,"ToeLFrPaw");  
  
    MMAccessor<HeadPointerMC>(headpointer_id)->  
        lookAtPoint(Ptarget(1),Ptarget(2),Ptarget(3),  
                    head_to_paw_distance);  
  
}
```



# General Inverse Kinematics

- Inverse kinematics solver included in PostureEngine.
- `solveLinkPosition(const NEWMAT::ColumnVector &Ptgt,  
                    unsigned int link,  
                    const NEWMAT::ColumnVector &Peff)`
  - Ptgt is the target point to move to
  - link is the index of some effector on the body
  - Peff is a point on the effector that is to be moved to Ptgt
- Returns true if a solution was found. False if no solution exists (e.g., joint limits exceeded, distance too far, etc.)
- Solution is stored in the PostureEngine as joint values.

# Classic Kinematics Application: Arm Control

Tekkotsu will soon have support for Lynx Motion arms:

