Outline

Kinematics is the study of how things move.

- Homogeneous coordinates
- Kinematic chains
  - Robots are described as collections of kinematic chains
- Reference frames
- Kinematics and PostureEngine classes
- Forward kinematics: calculating limb positions from joint angles. (Straightforward matrix multiply.)
- 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.
  - Points at infinite distance: last component is 0.

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

- Allows us to perform a variety of transformations using matrix multiplication:
  Rotation, Translation, Scaling

- Tekkotsu uses 3D coordinates (so 4-dimensional vectors) for everything.
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.

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

\[
T \vec{v} = \begin{bmatrix}
x \cos \theta + y \sin \theta + dx \\
-x \sin \theta + y \cos \theta + dy \\
z + dz \\
s
\end{bmatrix} = \begin{bmatrix}
(x \cos \theta + y \sin \theta + dx)/s \\
(-x \sin \theta + y \cos \theta + dy)/s \\
(z + dz)/s \\
1
\end{bmatrix}
\]
Transformations Are Composable

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

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

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

$$\text{RotateAbout}(p, \theta) = \text{Translate}(p) \cdot \text{Rotate}(\theta) \cdot \text{Translate}(-p)$$
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 instead of 6 because branched chains were formerly not supported:
  - 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).
Chiara Kinematic Chains

- The Chiara has 8 major kinematic chains:
  - Head / camera / IR
  - Arm
  - Left front leg
  - Right front leg (4-dof)
  - Left middle leg
  - Right middle leg
  - Left back leg
  - Right back leg

- Chains are defined in project/ms/config/Chiara.kin
  - Root Control > Framework Demos > Kinematics Demos > DisplayKinTree
Reference Frames

- Every link has an associated reference frame.
- Denavit-Hartenberg conventions: all links 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
Leg Reference Frames

ERS-7 Legs

<table>
<thead>
<tr>
<th></th>
<th>$\Delta x$</th>
<th>$\Delta y$</th>
<th>$\Delta z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. - shoulder</td>
<td>65</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2. - elevator</td>
<td>0</td>
<td>0</td>
<td>62.5</td>
</tr>
<tr>
<td>3. - knee</td>
<td>69.5</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>f4. - ball</td>
<td>69.987</td>
<td>-4.993</td>
<td>4.7</td>
</tr>
<tr>
<td>h4. - ball</td>
<td>67.681</td>
<td>-18.503</td>
<td>4.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\Delta x$</th>
<th>$\Delta y$</th>
<th>$\Delta z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. - shoulder</td>
<td>65</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2. - elevator</td>
<td>0</td>
<td>0</td>
<td>62.5</td>
</tr>
<tr>
<td>3. - knee</td>
<td>69.5</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>f4. - ball</td>
<td>69.987</td>
<td>-4.993</td>
<td>4.7</td>
</tr>
<tr>
<td>h4. - ball</td>
<td>67.681</td>
<td>-18.503</td>
<td>4.7</td>
</tr>
</tbody>
</table>
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
  - CameraFrameOffset
  - LFrLegOffset + ElevatorOffset

- Denavit-Hartenberg conventions specify how to express the relationship between one reference frame and the next: \( d, \theta, r, \alpha \).
Denavit-Hartenberg Video

http://www.youtube.com/watch?v=rA9tm0gTln8
DH Wizard

- Tool for editing kinematic descriptions. Outputs a kin file.
DH Wizard
Kinematics Class

- Tekkotsu contains its own kinematics engine for kinematics calculations, modeled after ROBOOP.
- The Kinematics class provides access to basic 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.
fmat

- Tekkotsu uses the fmat package to represent coordinates and transformation matrices.
- fmat is optimized for efficient representation of small, fixed-size matrices and vectors.

```cpp
fmat::Column<4> v, w;
v = fmat::pack(5.75, 30.0, 115, 1);
w = fmat::pack(17, -4.2f, 100, 1);

fmat::Matrix<4,4> T;
T = v * w.transpose();
```
fmat::Transform

• Transformation matrices using homogenous coordinates are $4 \times 4$.
• But the last row is always $[0 \ 0 \ 0 \ 1]$.
• So fmat eliminates the last row and overloads the arithmetic operators to make the math work correctly.
• fmat::Transform is really a Matrix<$3,4$>
Converting Between Reference Frames

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

- Conversion requires computing a transformation matrix:

  ```
  fmat::Transform linkToBase(unsigned int link)
  ```

  ```
  fmat::Transform baseToLink(unsigned int link)
  ```

  ```
  fmat::Transform linkTolink(unsigned int ilink, unsigned int olink)
  ```
Reference Frame Conversion 1

• Transform Base to Base:

```cpp
cmat::Transform T = kine->linkToBase(BaseFrameOffset);
cout << T.fmt("%8.3f") << endl;
```

• Result:

```
1.000  0.000  0.000
0.000  1.000  0.000
0.000  0.000  1.000
0.000  0.000  0.000
0.000  0.000  0.000
1.000  0.000  0.000
```
Reference Frame Conversion 2

Translate AIBO head tilt frame to base frame:

```cpp
const float headtilt = state->outputs[HeadOffset+TiltOffset];
cout << "Head tilt is " << headtilt * 180/M_PI << " degrees." << endl;

fmat::Transform TtiltL(kine->linkToBase (HeadOffset+TiltOffset));
cout << "tilt linkToBase=\n" << TtiltL.fmt("%8.3g") << endl;
```
At ~Zero Degree Tilt Angle

Head tilt is 1.25 degrees.

tilt linkToBase=

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.000</td>
<td>-0.022</td>
<td>0.000</td>
<td>67.500</td>
</tr>
<tr>
<td>1.</td>
<td>0.000</td>
<td>0.000</td>
<td>-1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2.</td>
<td>0.022</td>
<td>1.000</td>
<td>0.000</td>
<td>19.500</td>
</tr>
</tbody>
</table>

ERS-7 Head

<table>
<thead>
<tr>
<th></th>
<th>Δx</th>
<th>Δy</th>
<th>Δz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>67.5</td>
<td>0.0</td>
<td>19.5</td>
</tr>
<tr>
<td>2.</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3.</td>
<td>0.0</td>
<td>0.0</td>
<td>80</td>
</tr>
<tr>
<td>4.</td>
<td>40</td>
<td>-17.5</td>
<td>0</td>
</tr>
<tr>
<td>cam.</td>
<td>81.06</td>
<td>-14.6</td>
<td>0</td>
</tr>
<tr>
<td>IRn.</td>
<td>76.9</td>
<td>1.917</td>
<td>2.795</td>
</tr>
<tr>
<td>IRf.</td>
<td>76.9</td>
<td>1.052</td>
<td>-8.047</td>
</tr>
<tr>
<td>IRc.</td>
<td>109.136</td>
<td>-3.384</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ x_3 \angle x_4 = -23.6294^\circ \]
At ~ -30 Degree Tilt Angle

Head tilt is -29.5 degrees.

tilt linkToBase=

\[
\begin{array}{cccc}
0.871 & 0.492 & 0.000 & 67.500 \\
0.000 & 0.000 & -1.000 & 0.000 \\
-0.492 & 0.871 & 0.000 & 19.500 \\
\end{array}
\]

\[
\begin{align*}
\cos(-30^\circ) &= 0.866 \\
\sin(-30^\circ) &= 0.500
\end{align*}
\]
Interest Points

- Interest points on the head, legs, and body can be predefined for use in kinematics calculations.
- Not yet supported in new kinematics engine.

Interest Points:
A - LowerLeftLowerLip
B - LowerRightLowerLip
C - UpperLeftLowerLip
D - UpperRightLowerLip
E - LowerLeftUpperLip
F - LowerRightUpperLip
G - LowerLeftSnout
H - LowerRightSnout
I - UpperLeftSnout
J - UpperRightSnout
K - LeftMicrophone
L - RightMicrophone
M - HeadButton
N - XIRn
O - YIRn
P - ZIRn
Q - XIRf
R - YIRf
S - ZIRf
T - Xcam
U - Ycam
V - Zcam
Leg Interest Points

Interest Points:
A - Toe\{L,R\}\{Fr,Bk\}\text{Paw}_4
B - Lower\{Inner,Outer\}\text{Front}\{L,R\}\{Fr,Bk\}\text{Shin}_3
C - Lower\{Inner,Outer\}\text{Middle}\{L,R\}\{Fr,Bk\}\text{Shin}_3
D - Lower\{Inner,Outer\}\text{Back}\{L,R\}\{Fr,Bk\}\text{Shin}_3
E - Middle\{Inner,Outer\}\text{Middle}\{L,R\}\{Fr,Bk\}\text{Shin}_3
F - Upper\{Inner,Outer\}\text{Front}\{L,R\}\{Fr,Bk\}\text{Shin}_3
G - Upper\{Inner,Outer\}\text{Back}\{L,R\}\{Fr,Bk\}\text{Shin}_3
H - Lower\{Inner,Outer\}\text{Front}\{L,R\}\{Fr,Bk\}\text{Thigh}_2
I - Lower\{Inner,Outer\}\text{Back}\{L,R\}\{Fr,Bk\}\text{Thigh}_2
J - Upper\{Inner,Outer\}\text{Front}\{L,R\}\{Fr,Bk\}\text{Thigh}_2
K - Upper\{Inner,Outer\}\text{Back}\{L,R\}\{Fr,Bk\}\text{Thigh}_2
L - Upper\{L,R\}\text{Chest}_0
M - Lower\{L,R\}\text{Chest}_0
N - \{L,R\}\{Fr,Bk\}\text{Belly}_0
O - Lower\{L,R\}\text{Rump}_0
P - Upper\{L,R\}\text{Rump}_0

ERS-7 Legs

\begin{tabular}{lll}
\text{#} & \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 \\
\end{tabular}

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:

  ```cpp
  void getInterestPoint(const std::string &name,
                        unsigned int &link,
                        fmat::Column<4> &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:

  ```cpp
  fmat::Column<4>
  getInterestPoint(unsigned int link,
                   const std::string &name)
  ```
Forward Kinematics: Measure Distance From RFr Leg to Gripper

```cpp
$nodemethod doEvent {

    fmat::Transform rfrFoot =
        kine->linkToBase(FootFrameOffset+RFrLegOrder);
    fmat::Column<3> rfrFootPos = rfroot.translation();

    fmat::Transform gripper =
        kine->linkToBase(GripperFrameOffset);
    fmat::Column<3> gripperPos = gripper.translation();

    float dist = (rfrFootPos-gripperPos).norm();

    cout << "Distance is " << setw(5) < dist << " mm." << 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)
  - point described in base frame coordinates
  - calculates head joint angles
CameraTrackGripper Demo

Root Control > Framework Demos > Kinematics Demos > CameraTrackGripper

```cpp
$nodeclass CameraTrackGripper : StateNode : armRelaxer(), headMover() {
    MotionPtr<PIDMC> armRelaxer;
    MotionPtr<HeadPointerMC> headMover;

    $nodemethod doStart {
        addMotion(armRelaxer);
        addMotion(headMover);
        erouter->addListener(this,EventBase::sensorEGID);
    }
```
TrackGripper Behavior 2

```cpp
$nodemethod doEvent {
    fmat::Column<3> Pgripper =
        kine->linkToBase(GripperFrameOffset).translation();
    cout << "Transform:" << Tgripper.fmt("%8.3f") << endl;
    headMover->lookAtPoint(Pgripper[0],
        Pgripper[1],
        Pgripper[2]);
}
```
General Inverse Kinematics

• Inverse kinematics solver included in PostureEngine:

\[
\text{solveLinkPosition}(\text{const fmat::Column<3>& Ptgt,} \\
\text{unsigned int link,} \\
\text{const fmat::Column<3>& Peff})
\]

- Ptgt is the target point to move to (in base frame coordinates)
- link is the index of some effector on the body, e.g., ArmOffset+GripperOffset
- Peff is a point on the effector that is to be moved to Ptgt, in the reference frame of that effector.

• 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.
Solving the 1-Link Arm

Reachable if: \( L_1 = \sqrt{x^2 + y^2} \)

Solution: \( \theta_1 = \text{atan2}(y, x) \)
Solving the 2-Link Planar Arm

Target \((x,y)\)

\[
\begin{align*}
c_2 &= \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2} \\
s_2^+ &= \sqrt{1 - c_2^2} \\
\theta_2^+ &= \text{atan2}(s_2^+, c_2) \\
K_1 &= L_1 + c_2L_2 \\
K_2 &= s_2^+L_2 \\
\theta_1 &= \text{atan2}(y, x) - \text{atan2}(K_2, K_1)
\end{align*}
\]

Reachable if: \(c_2^2 \leq 1\)
Two Possible Solutions

\[
\begin{align*}
  s^+_2 &= \sqrt{1-c^2_2} \\
  \theta^+_2 &= \text{atan2}(s^+_2, c_2)
\end{align*}
\]

\[
\begin{align*}
  s^-_2 &= -\sqrt{1-c^2_2} \\
  \theta^-_2 &= \text{atan2}(s^-_2, c_2)
\end{align*}
\]
Solving the 3-Link Planar Arm

- Choose tool angle $\phi$
- Given target position $x_t$, $y_t$, calculate wrist position: $x_w$ and $y_w$
- Solve 2-link problem to put wrist at $x_w$, $y_w$. 

Target $(x,y)$
Customized Kinematics Solvers

• For some simple kinematic chains, such as a pan/tilt, we can write analytical solutions to the IK problem.

• For the general case, must use gradient descent search.

See IK videos.