Outline

Kinematics is the study of how things move.

- Kinematic chains
  - Robots are described as collections of kinematic chains
- Reference frames
- Homogeneous coordinates
- 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.)
Robots As Kinematic Chains

- Tekkotsu allows branching chains, so robots are trees.
- The root of the tree is called the *BaseFrame* in Tekkotsu.
- It is typically at the center of the robot's body.
Chains = Joints + Links

- A chain is a 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 rotational joint has a rotation transform; each link has a translation transform.

- The math for this will be shown later in this lecture.
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
Calliope Kinematic Chains

**BaseFrame**
- center of axle
- WHEEL:L, WHEEL:R

**NECK:**
- PAN
- TILT

**CameraFrame**
- ARM:base
- ARM:shoulder
- ARM:elbow
- ARM:wrist
- ARM:wristrot

**GripperFrame**
- ARM:gripperleft

**LeftFingerFrame**
- ARM:gripperright

**RightFingerFrame**

Use the DisplayKinTree demo to show the kinematic tree of the robot.

Root Control
- Framework Demos
  - Kinematics Demos
    - DisplayKinTree
Reference Frames

- Every joint has an associated reference frame.
- Additional reference frames for camera, toes, etc.

- Denavit-Hartenberg conventions: joints rotate about their z-axes.
- The x and y axes follow the right hand rule.
Chain of Reference Frames

- **BaseFrame:** $z$ is up, $x$ is forward, $y$ is left.
  - This convention is also used for localShS and worldShS.

- **Axis of rotation** determines $z$ for a joint.

- **The head chain:**
  - Base frame $0$ \( z_0 = \text{“up”} \)
  - Tilt joint $1$ \( y_1 = \text{“up”} \)
  - Pan joint $2$
  - Nod joint $3$
  - Camera $4$ \( z_4 = \text{“out”}, \ x_4, y_4 = \text{image plane} \)
Reference Frame Naming Conventions

- Use the same offset-based indexing scheme as for joint names in motion commands and world state vectors:
  - BaseFrameOffset
  - HeadOffset+TiltOffset, HeadOffset+PanOffset
  - CameraFrameOffset
  - ArmShoulderOffset, ArmElbowOffset, ArmWristOffset, etc.
  - GripperFrameOffset

- Denavit-Hartenberg conventions specify how to express the relationship between one reference frame and the next: d, θ, r, α.
Denavit-Hartenberg Video

http://www.youtube.com/watch?v=rA9tm0gTln8
Summary of D-H Conventions

1) Move by $d$ along $z_{n-1}$

2) Rotate by $\theta$ around $z_{n-1}$

3) Move by $r$ along $x_n$, which is the common normal of $z_{n-1}$ and $z_n$

4) Rotate by $\alpha$ along $x_n$

When $z_{n-1}$ and $z_n$ are parallel:
- $d$ is arbitrary
- $\alpha$ is 0
The Tekkotsu .kin File

- See project/ms/config/Calliope5KP.kin

- Contains four types of information:
  - Kinematic description of the robot following D-H conventions, used by Tekkotsu's kinematics solvers.
  - Additional joint and link information, such as min, max, and offset values, mass, center of mass, etc.
  - Paths to mesh files (models) for selected joints, used by Mirage to render the robot.
  - Collision models for selected components, used by Mirage to determine how the robot interacts with the world.
DH Wizard

- Tool for editing kinematic descriptions. Outputs a kin file.
DH Wizard
DH Wizard
Now, The Math...

- How do we represent transformations from one reference frame to the next in a kinematic chain?
  - Homogeneous coordinates
  - Transformation matrices

- How do we perform these calculations in C++?
  - The fmat package

- How do I get Tekkotsu to do the work for me?
  - Forward kinematics solver
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.

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

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

\[
\text{RotateAbout}(p, \theta) = \text{Translate}(p) \cdot \text{Rotate}(\theta) \cdot \text{Translate}(-p)
\]
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>
The 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.
- Defined in Tekkotsu/Motion/Kinematics.h
- 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.
Converting Between Reference Frames

- Most common conversions are between the base frame (body coordinates) and a limb or camera frame.
- Conversion requires computing a transformation matrix.
- Specify the frame with an unsigned int (a joint offset).

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

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

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

• Transform Base to Base:

```cpp
fmat::Transform T = kine->linkToBase(BaseFrameOffset);
cout << T.fmt("%8.3f") << 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
```
Translate Calliope head pan frame to base frame:

```cpp
const float headpan = state->outputs[HeadOffset+PanOffset];
cout << "Head pan is \n" << headpan * 180/M_PI 
     << " degrees." \n     << endl;

fmat::Transform TpanL = kine->linkToBase(HeadOffset+PanOffset);
cout << "pan linkToBase=\n" << TPanL.fmt("%8.3f") \n     << endl;
```
At ~Zero Degree Pan Angle

Head pan is 0.0016182 degrees.

\[
\text{pan linkToBase} = \\
\begin{bmatrix}
1.000 & -0.000 & 0.000 & 75.230 \\
0.000 & 1.000 & 0.000 & 0.000 \\
0.000 & 0.000 & 1.000 & 383.916 \\
0.000 & 0.000 & 0.000 & 1.000
\end{bmatrix}
\]
At ~ 30 Degree Pan Angle

Head pan is 32.7 degrees.

\[
\text{pan linkToBase} = \\
\begin{bmatrix}
0.846 & -0.534 & 0.000 & 75.230 \\
0.534 & 0.846 & -1.000 & 0.000 \\
0.000 & 0.000 & 0.000 & 383.916
\end{bmatrix}
\]

\[
\cos(30^\circ) = 0.866 \\
\sin(30^\circ) = 0.500
\]
How About Tilt w/Head Centered?

Head pan is -0.001547 degrees.

pan linkToBase =
[ 1.000  -0.000  0.000  75.230
  0.000   1.000  0.000   0.000
  0.000   0.000   1.000  383.916 ]

Head tilt is 0.009223 degrees.

tilt linkToBase =
[ 1.000   -0.000   -0.000   97.730
 -0.000   -0.000   1.000  -0.001
  0.000   1.000   -0.000  422.916 ]
Forward Kinematics: Measure Distance From Wrist to Arm Base

```cpp
$nodeclass ComputeDistance : StateNode : doStart {

    fmat::Transform wrist =
        kine->linkToBase(ArmWristOffset);
    fmat::Column<3> wristPos = wrist.translation();

    fmat::Transform armbase =
        kine->linkToBase(ArmBaseOffset);
    fmat::Column<3> armbasePos = armbase.translation();

    float dist = (wristPos-armbasePos).norm();

    cout << "Distance is " << setw(5) < dist << " mm." << endl;
}

startnode: ComputeDistance =T(1000)=> startnode
```
Inverse Kinematics

- 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
$nodeclass CameraTrackGripper : StateNode : {

$nodeclass HeadMover : HeadPointerNode : doStart {
    fmat::Transform Tgripper =
        kine->linkToBase(GripperFrameOffset);

    fmat::Column<3> Pgripper = Tgripper.translation();

    std::cout << "Transform:\n"
        << Tgripper.fmt("%8.3f") << std::endl;

    getMC()->lookAtPoint(Pgripper[0], Pgripper[1], Pgripper[2]);
}
virtual void setup() {
    MotionManager::MC_ID headmc =
    addMotion(MotionPtr<HeadPointerMC>());

    $statemachine{
    startnode: StateNode =N=> {headmover, unrelaxed}
    headmover: HeadMover[setMC(headmc)] =E(sensorEGID)=> headmover
    unrelaxed: SpeechNode("arm not relaxed")
      =B(GreenButOffset)=> armrelaxer
    armrelaxer: SpeechNode("arm is relaxed")
      =N=> PIDNode(ArmOffset, ArmOffset+NumArmJoints, 0.f)
      =B(GreenButOffset)=> unrelaxed
    }
}
Solving the 1-Link Arm

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

Solution: \( \theta_1 = \text{atan2}(y, x) \)
Consider a 2-link arm, with joint constraints

\[ 0^\circ < \theta_0 < 90^\circ, \quad -90^\circ < \theta_1 < 90^\circ \]

**Configuration Space**: robot’s internal state space (e.g. joint angles)

**Work Space**: set of all possible end-effector positions
Solving the 2-Link Planar Arm

Target \((x, y)\)

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

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

\[ s_2^- = -\sqrt{1-c_2^2} \]
\[ \theta_2^- = \text{atan2}(s_2^-, c_2) \]

“Elbow up”

\[ s_2^+ = \sqrt{1-c_2^2} \]
\[ \theta_2^+ = \text{atan2}(s_2^+, c_2) \]

“Elbow down”
How Many Degrees of Freedom Are Enough?

- With 2 dof you can put the end effector at any point in the workspace.
- But you can't control end-effector orientation.
  - What if the arm is holding a screwdriver?
- With 3 dof in the same plane you can control both position and orientation.
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 $x_w$, $y_w$.

If you don't know $\phi$, pick an arbitrary value and search from there until you find a solution that works.
Towers of Hanoi in the Plane

Video by Michel Brudzinski and Evan Patton at RPI.
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.
Inverse Kinematics Functions

• Inverse kinematics solver included in PostureEngine:

```cpp
solveLinkPosition(const fmat::Column<3> &Ptgt,
                 unsigned int link,
                 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., 
    GripperFrameOffset
  – Peff is a point on the effector that is to be moved to Ptgt, in
    the reference fame 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.
$\textit{GripperTrackCamera}$

```c
$\texttt{nodeclass GripperTrackCamera : StateNode \{}

$\texttt{nodeclass ArmMover : PostureNode : doStart \{}

  \texttt{fmat::Column<3> targetInCam = fmat::pack(0, 0, 100);
  fmat::Column<3> targetInBase =
      kine->linkToBase(CameraFrameOffset) * targetInCam;
  fmat::Column<3> noOffset = fmat::pack(0, 0, 0);

  getMC()->solveLinkPosition(targetInBase,
      LeftFingerFrameOffset, noOffset);

\}}$
```
GripperTrackCamera (2)

virtual void setup() {
    MotionManager::MC_ID armmc =
        addMotion(MotionPtr<PostureMC>());

    $statemachine{
        startnode: ArmMover[setMC(armmc)]
            =E(sensorEGID)=> startnode

    }
}


Additional IK Functions

PostureEngine provides:

- `solveLinkPosition(...)`
- `solveLinkVector(...)`
- `solveLinkOrientation(...)`
- `solveLink(...)`

The actual IK calculations for Calliope are done in Tekkotsu/Motion/IKCalliope.cc
Calliope's 5-dof ARM

- Only one degree of freedom in the horizontal plane:
  - ARM:base

- Three degrees of freedom in a vertical plane:
  - ARM:shoulder, ARM:elbow, ARM:wrist

- An additional degree of freedom in an orthogonal plane:
  - ARM:wristrot

- Conclusion: can only partially control the 3D pose of the end-effector.
  - What kinds of motions can this arm not make?