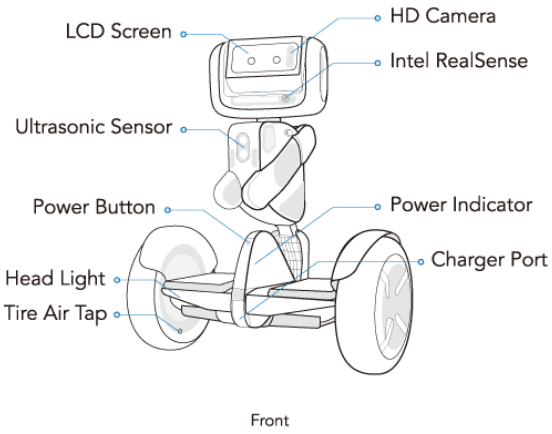
Loomo SDK 中文文档

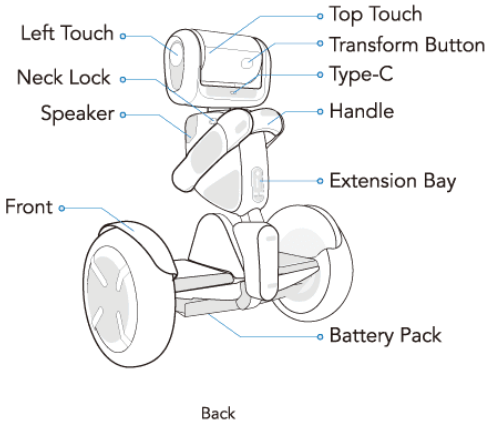
注：本文档资料来源 Loomo SDK，仅供参考学习使用

Contact Me:
Email：tangjunjie98@foxmail.com
GitHub：https://github.com/tjj1998

目录

Loomo 介绍





Hardware Specs	Descriptions
Dimension	Height 0.64m, length 0.57m, width 0.28m
Weight	17.5kg, including the battery pack
Battery Capacity	310Wh
Speed Limit	8 kilometers per hour (software limit for the Alpha edition)
Pan Tilt Unit Range	Pan ±150 degrees, Tilt -90 ~ +180, zero is horizontal facing front
LCD Screen	4.3 inch

Platform Specs	Descriptions
Processor	Intel Atom Z8750, 4 cores, 2.4GHz, x86-64 architecture
Operating System	Customized system based on Android 5.1
Memory	4 GB
Storage	64 GB
USB port	USB 3.0 Type-C cable (compatible with USB 2.0)
Extension Bay	USB 2.0 and 24 voltage power (1A limit)

Sensor Specs	Descriptions
RealSense	Real time 30Hz RGB-Depth streaming for both indoor and outdoor uses
HD Camera	High resolution with 104° FOV (Pending fine turning. Image quality is not optimal)
Mic Array	Beamforming and voice localization, powered by 5 microphones
Ultrasonic Sensors	Obstacle distance calculation
Touch Sensors	Located at robot head left, right & back
Wheel Odometry	Approximately 4 degrees precision
Base IMU	6-axis IMU

补充：
在自平衡车辆(SBV)模式，最高时速 5m/s(18km/h)，在负重 75kg 的情况下可运行 30KM；
在机器人模式，续航时间依赖于 CPU 使用情况，平均为 6-8 小时

Loomo SDK

一、 注意事项

1. 在使用 SDK 功能前，需要调用 bindService()，在使用结束后，调用 unbindService()
2. Vision SDK 提供的时间戳为 RealSense 的时间戳，非系统的时间戳。系统的时间戳可以通过 getTimeStamp()来获取。
3. 后端服务受限，Loomo 只支持前端运行的服务，当应用置于后端时，SDK 会自动断开服务。需要运行在前端的服务有：Vision，Speech，Locomotion，Interaction

二、 机器人事件广播

PowerEvent：

1. BATTERY_CHANGED 电池等级改变广播
2. POWER_DOWN 休眠关机广播
3. POWER_BUTTON_PRESSED 电源按钮按下广播
4. POWER_BUTTON_RELEASED 电源按钮释放广播

TransformEvent：

1. SBV_MODE 进入 SBV 模式广播
2. ROBOT_MODE 进入机器人模式广播

HeadEvent：

1. PITCH_LOCK 俯仰角锁定广播，此时 Loomo 面向下并锁定，发生于进入 SBV 模式
2. PITCH_UNLOCK 俯仰角解锁广播，Loomo 头部俯仰角解锁，发生于进入 Robot 模式
3. YAW_LOCK 偏航角锁定广播，此时 Loomo 面向左并锁定，发生于进入 SBV 模式
4. YAW_UNLOCK 偏航角锁定广播，Loomo 头部偏航角解锁，发生于进入 Robot 模式

ActionEvent：

1. STEP_ON 踏板承重广播，当压力传感器检测到一定压力时广播
2. STEP_OFF 踏板失重广播，当压力传感器检测到压力消失时广播
3. LIFT_UP Loomo 离地广播 当 Loomo 脱离地面时广播
4. PUT_DOWN Loomo 置地广播，当 Loomo 放到地面时广播，松开手柄内部触发此事件。
5. PUSHING 当用户触碰头部按钮时广播，此时 Loomo 的运动完全由人控制机器人头部，任何运动相关 API 会被忽视。
6. PUSH_RELEASE 当用户不在触碰头部按钮时广播，Loomo 的运动将回到之前的状态。

BaseEvent：

1. BASE_LOCK 基座锁定广播，车轮里程计断电，Loomo 不再自平衡。必须放置在地面上，这通常发生于 OTA 更新。
2. BASE_UNLOCK 基座解锁广播，车轮里程计重新上电，用户可以让机器人站立或自平衡，这通常发生于 OTA 更新后。
3. STAND_UP 这条信息表明机器人的基座是直立的。

三、 VISION 模块 (Detailed SDK DOC)

:视觉服务用于控制 Intel RealSense 摄像头的初始化和配置，允许开发人员检索实时图像数据。支持 2 百万像素的 RGB、红外和 3D 深度图像。

初始化视觉实例并绑定到服务:

```
mVision = Vision.getInstance();
mVision.bindService(this, mBindStateListener);
```

因为视觉服务不能设置捕获的图像参数，例如图像大小、像素格式、帧速率等，所以应用程序需要请求这些参数(并适配这些参数)，以便在 Loomo 的屏幕上显示图像。因此，绑定到 vision 服务后，键入以下代码::

```
StreamInfo[] infos = mVision.getActivatedStreamInfo();
```

图像输出格式:

Image Type	Resolution	Pixel format
Depth Image	320x240	Z16
Color Image	640x480	ARGB8888

获取图像:

```
private Bitmap mBitmap = Bitmap.createBitmap(640, 480, Bitmap.Config.ARGB_8888);

mVision.startListenFrame(StreamType.DEPTH, new Vision.FrameListener() {
    @Override
    public void onNewFrame(int streamType, Frame frame) {
        mBitmap.copyPixelsFromBuffer(frame.getByteBuffer());
    }
});
```

鱼眼相机(Fisheye Camera)

Loomo 配备了广角 FOV 鱼眼相机，可以实现视觉 SLAM。目前，鱼眼相机还没有公开向公众开放，如果使用需要申请授权， e-mail service@loomo.com。
在申请授权后通过一下代码获取鱼眼相机图像:

```
private Bitmap mFishEyeBitmap = Bitmap.createBitmap(640, 480, Bitmap.Config.ALPHA_8);

mVision.startListenFrame(StreamType.FISH_EYE, new Vision.FrameListener() {
    @Override
    public void onNewFrame(int streamType, Frame frame) {
        mFishEyeBitmap.copyPixelsFromBuffer(frame.getByteBuffer());
    }
});
```

检测跟踪系统(Detection-tracking system，DTS)

检测跟踪系统包括物体检测和跟踪。同时使用 DTS 和其他 loomo 提供的服务，可以轻松地设计和实现人员跟踪和跟随应用程序。

注意：DTS 有以下已知问题: 当跟踪某人时，在某些情况下，onPersonTracked 回调函数返回的与 Person 实例的距离有误。最佳的距离是在 0.35 米到 5 米之间。

为了使用 DTS，首先需要获取 Vision 实例并绑定到 VisionService 。接着，从 Vision 获取 DTS 实例。在检测或跟踪前，需要选择视频流模式。在相机模式，VisionService 将打开和管理平台摄像机。通过调用 start()方法，DTS 模块将开始工作。当你不需要使用 DTS 时，调用 stop()方法。以下代码描述如何使用 DTS:

```
// get the DTS instance
DTS dts = mVision.getDTS();

// set video source
dts.setVideoSource(DTS.VideoSource.CAMERA);

// set preview surface
Surface surface = new Surface(autoDrawable.getPreview().getSurfaceTexture());
dts.setPreviewDisplay(surface);

// start dts module
dts.start();

// detect person in 3 seconds
Person[] persons = dts.detectPersons(3 * 1000 * 1000)

// track the first person(if there is a person detected)
dts.startPersonTracking(persons[0], 10 * 1000, new PersonTrackingListener {...})

// stop person tracking
dts.stopPersonTracking()

// stop dts module
dts.stop();
```

如果将 null 传递给 startPersonTracking, DTS 将自动尝试在平台图像中心找到一个人。避障 API 也在同时工作，但机器人有时会碰到人或其他障碍。在测试 FollowMe 样例时，要始终保持警惕。唯一的区别是开始方法:

```
// Loomo will detect obstacles and avoid them
dts.startPlannerPersonTracking(null, mPersonTrackingProfile, 60 * 1000 * 1000, new PersonTrackingWithPlannerListener{...});
```

四、 SPEECH 模块 (Detailed SDK DOC)

Loomo 配备了先进的 5 声道麦克风阵列，可以定位声源，同时提供语音唤醒和高精度语音识别功能。SPEECH 分为两个模块：：

Recognition 模块: 负责语音唤醒、语音识别、定制语法。您可以配置 GrammarConstraint 以识别不同场景中的不同内容。除了语音识别，该识别服务还支持波形语音记录，可以远距离记录语音，并有效降低背景噪声。

Speaker 模块: 负责文本到语音(**text-to-speech**)的转换。

识别模块(Recognition)

要激活 Loomo 的语音识别服务，需要使用诸如“Ok Loomo”、“Hello Loomo”、“Loomo Loomo”、“Loomo transporter”、“Hi Loomo”和“Hey Loomo”这样的唤醒词来唤醒设备。

为了创建有意义的语音命令，你需要为 Loomo 创建语音短语/句子。在 Recognition SDK 中，使用 Slot 和 GrammarConstraint 帮助 Loomo 理解语音命令:

Slot: 一个字或一组字。

GrammarConstraint: 接受一个或多个 **Slot** 的语言语法。每个 **Slot** 中的单词可以组合成一个句子。

以下是一个例子:

```
GrammarConstraint: <Slot1> [<Slot2>] <Slot3>
Slot1: turn/walk
Slot2: to the
Slot3: left/right
```

注意：在这个样例中，Slot2 为可选内容;

注意：如果您添加了两个以上的 GrammarConstraints，那么只有最新的两个可以正常工作，而且由于初始化非常耗时，所以有数量限制。

在上述样例中，以下语句(状态)可以被识别:

- turn left
- turn right
- turn to the left
- turn to the right
- walk left
- walk right
- walk to the left
- walk to the right

在 Recognition 服务中，可以使用 RecognitionListener 中的 onRecognitionResult 回调函数和 onRecognitionError 回调函数的返回值进行连续识别。这种设计使开发人员可以用新的命令替换之前识别的命令，实现准确的连续识别。

以下为 Recognizer 使用样例介绍：

在使用前，需要初始化和绑定到 Recognizer 服务:

```
mRecognizer = Recognizer.getInstance();

//bind the recognition service.
mRecognizer.bindService(MainActivity.this, mRecognitionBindStateListener);
```

获取当前语言：

```
mRecognitionLanguage = mRecognizer.getLanguage();
```

注意：目前版本只支持普通话和英语。请注意，混合语句无法被识别。语言类型通过系统语言设置来决定。

添加预先生成的 GrammarConstraint：

```
mRecognizer.addGrammarConstraint(mTwoSlotGrammar);
```

开始唤醒和识别，传递 WakeupListener 实例和 RecognitionListener 实例，获得唤醒和识别状态:

```
mRecognizer.startRecognition(mWakeupListener, mRecognitionListener);
```

WakeupListener and RecognitionListener 更多信息详见样例代码和 SDK。

注意：Loomo 保留了一些不能作为识别词使用的词：

"loomo loomo", "ok loomo", "hey loomo", "yo loomo", "hello loomo", "hi loomo", "come here", "come to me", "open camera", "turn on the camera", "close camera", "turn off the camera", "follow me", "stop follow me", "stop following"

语音模块(Speaker)

Speaker 负责将文本转换为语音。在使用前，需要初始化 Speaker 实例并绑定到服务:

```
mSpeaker = Speaker.getInstance();

//bind the speaker service.
mSpeaker.bindService(MainActivity.this, mSpeakerBindStateListener);
```

获取当前语言：

```
mSpeakerLanguage = mSpeaker.getLanguage();
```

注意：目前版本只支持普通话和英语。请注意，混合语句无法被识别。

设置文本到语音转换的音量，有效值[0,100]:：

```
mSpeaker.setVolume(50);
```

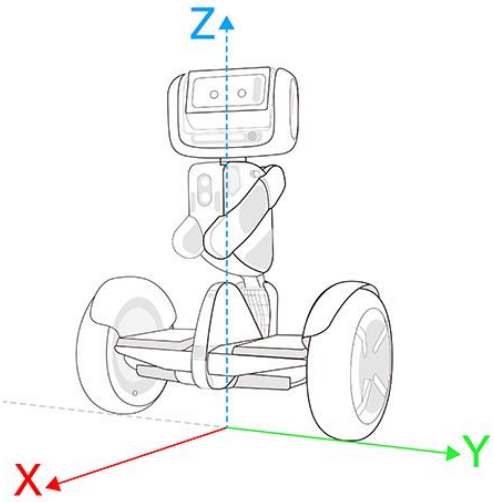
通过 Speaker 服务进行 Speak，通过 TtsListener 设置获取当前 Speak 的进度：

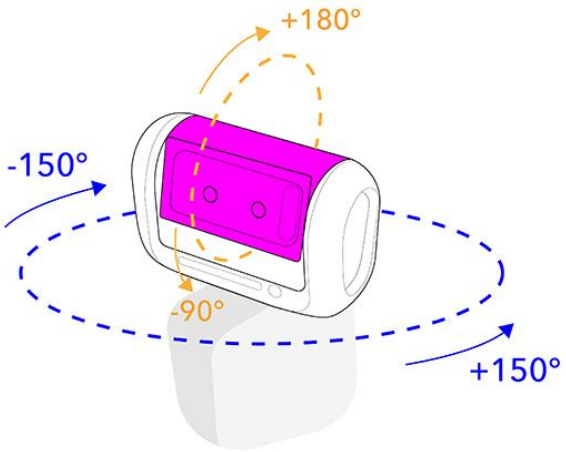
```
mSpeaker.speak("hello world, I am a Segway robot.", mTtsListener);
```

：

五、 Locomotion 模块 (Detailed SDK DOC For Head/Base)

你可以使用 Locomotion 服务控制 Loomo 的头和基座。为了控制头部和基座，您需要了解 Loomo 的坐标系统。下面是机器人的坐标系定义：





头部(Head)

在 Loomo 的头部，有两个维度可以控制:俯仰角和偏航角，支持**平滑跟踪模式**和**锁定模式**:
平滑跟踪模式: 在这种模式下，头部的俯仰轴是稳定的，可以有效的过滤身体的冲击。偏航轴可以随着底座旋转。在这种状态下，可以以底座为参照系设置角度来控制头部。
锁定模式: 在这种模式下，俯仰轴在头部是稳定的，可以有效的过滤身体的冲击。偏航轴在世界坐标系中指向某个方向。如果设置为固定点，则利用该模型可以获得稳定的拍摄效果。在此模式下，您可以通过设置头部旋转速度来控制头部方向。
表情模式: 此模式仅在表情 SDK 中使用。

以下是样例代码说明：
在使用前，你需要初始化 Head 实例，并绑定到服务：

```
// bindService, if not implemented, the Head API will not work.
mHead.bindService(getApplicationContext(), mServiceBindListener);
```

设置为平滑跟踪模式，控制头部位置：

```
mHead.setMode(Head.MODE_SMOOTH_TACKING);

mHead.setWorldPitch((float) Math.PI);
mHead.setWorldYaw((float) Math.PI);
```

设置为锁定模式，控制头部的速度：

```
mHead.setMode(Head.MODE_ORIENTATION_LOCK);

mHead.setYawAngularVelocity((float) Math.PI);
mHead.setPitchAngularVelocity((float) Math.PI);
```

增加了一个新方法用于改变头部灯光模式：

```
mHead.setHeadLightMode(0);
```

注意：在接受闪烁命令之前，必须发送另一个命令。

注意：该值的取值范围为[0，13]，默认值为 0。

底座(Base)

Loomo 的底座采用了最新的两轮自平衡技术。通过使用 Base SDK，开发人员可以控制底座的线速度和角速度。在控制机器人底座时，需要将其设置为连续的速度。如果在 700 毫秒内没有速度设置，机器人底座将停止移动。

在使用它之前，你需要初始化 Base 实例，并绑定到服务：

```
mBase.bindService(getApplicationContext(),mServiceBindListener);
```

Odometry

里程计提供了相对于起始位姿的一个大致的机器人位姿(x, y，orientation)。坐标系的原点是基底的 x, y, z 中心。x 轴指向机器人的前方，y 轴指向左边，z 轴垂直于地面，遵循右手定则(逆时针)。它每 50 毫秒更新一次。对于校准良好的机器人，里程计精度为 99% - 99.9%。

调用 getOdometryPose()获取特定时间(毫秒)的位姿。我们对于里程计数据只缓冲几秒钟。

```
mBase.getOdometryPose(System.currentTimeMillseconds() * 1000);
```

如果你想获取最新的位姿，可以输入-1 到该 API：

```
mBase.getOdometryPose(-1);
```

设置 Loomo 的起始点

我们提供了一个 API 来设置起始点。注意，一旦设置了起始点，getOdometryPose()将根据它之前的起始点返回 Pose2D。因此在再次设置起始点之前，请确保调用 cleanOriginalPoint()来清除前面的起始点。

```
mBase.cleanOriginalPoint();
Pose2D pose2D = mBase.getOdometryPose(-1);
mBase.setOriginalPoint(pose2D);

mBase.addCheckPoint(1f, 0, (float) (Math.PI /2));
mBase.addCheckPoint(1f, 1f, (float) (Math.PI));
mBase.addCheckPoint(0f, 1f, (float) (-Math.PI /2));
mBase.addCheckPoint(0, 0, 0);
```

setOriginalPoint()可以通过任何 Pose2D 数据来设置车轮里程计，并从该点开始计算。

底盘控制模式(Base Control Modes)

Loomo 底座有三种不同的模式：

- CONTROL_MODE_RAW: 在该模式下，你可以通过设置线速度和角速度直接操纵 Loomo 的底座；

- CONTROL_MODE_NAVIGATION:在该模式下，你可以通过添加 checkpoints(检查点)的方式让 Loomo 运动到某一固定点。Loomo 将尝试依次到达设置的每个检查点。该精度大约为 0.25m。
- CONTROL_MODE_FOLLOW_TARGET: 这种模式是为“follow me”场景设计的。在该模式下，Loomo 将通过接收相对于自身的距离和方向来尝试移动到某一点。addCheckPoint() API 无法使用，可以通过调用 updateTarget 替代。updateTarget() API 将使机器人移动到最新点。

Loomo 控制注意事项：

1. Loomo 只有两个轮子，因此它像所有的 Segways 机器人一样有局限性:“保持平衡是它的首要任务”。因为 Loomo 总是需要保持自平衡，所以当它想要加速时，它会先倾斜身体，然后移动。相应地，当 Loomo 想要减速时，它会先将身体向相反的方向倾斜。因此，在控制 Loomo 时总是存在延迟。
2. 在 RAW 模式下，最大允许的速度是 0.35m/s。但在 Navigation 模式下，Loomo 的速度可以设置得更快。
3. Navigation 模式允许 Loomo 轻松地移动到某个确定点，精确度约为 0.25 米。所以如果把一个点设置在离 Loomo 不到 0.25 米的地方，它就不会移动。

CONTROL_MODE_RAW 样例代码：

设置机器人当前的线速度和角速度:

```
mBase.setControlMode(Base.CONTROL_MODE_RAW);
mBase.setLinearVelocity(1.0f);
mBase.setAngularVelocity(0.15f);
```

CONTROL MODE NAVIGATION 样例代码：

在该模式下添加一些检查点:

```
mBase.setControlMode(Base.CONTROL_MODE_NAVIGATION);
mBase.setOnCheckPointArrivedListener(new CheckPointStateListener() {
    @Override
    public void onCheckPointArrived(CheckPoint checkPoint, final Pose2D realPose, boolean isLast) {
    }
    @Override
    public void onCheckPointMiss(CheckPoint checkPoint, Pose2D realPose, boolean isLast, int reason) {
    }
});

mBase.addCheckPoint(1f, 0, (float) (Math.PI /2));
mBase.addCheckPoint(1f, 1f, (float) (Math.PI));
mBase.addCheckPoint(0f, 1f, (float) (-Math.PI /2));
mBase.addCheckPoint(0, 0, 0);
```

如果你想在机器人走完所有检查点之前停止机器人，可以将 Base 设置为 CONTROL MODE RAW 模式。

注意:

1.Loomo 不能横着走。如果目标点在 Loomo 旁边，它需要走一个弧线到达该目标点。

2.跟开车一样，Loomo 在转弯时会减速，这会由 SDK 内部处理。可以使用以下方法:

```
public CheckPoint addCheckPoint(float x, float y)
```

如果你想要一个弯曲的路径转弯，使用一下方法:

```
public CheckPoint addCheckPoint(float x, float y, float theta)
```

Loomo 将到达点(x,y)并将方向转为 theta。(theta 取值范围为[-π，π])。

3. 目前该模式有个已知 bug。onCheckPointArrived()回调函数应该在 Loomo 到达每个检查点时调用。但是，现有的一个 bug 使得 Loomo 只有在到达最后一个检查点时才会被调用。

CONTROL_MODE_FOLLOW_TARGET 样例代码：

在该模式下让机器人跟随某人:

```
mBase.setControlMode(Base.CONTROL_MODE_FOLLOW_TARGET);

while(someConditionIsTrue) {
    Person person = getPersonFromDTS();
    if (person.getDistance() > 0.35 && person.getDistance() < 5) {
        float followDistance = (float) (personDistance - 1.2);
        float theta = person.getTheta();
        mBase.updateTarget(followDistance, theta);
    }
}
```

注意：

1. updateTarget()必须持续调用。如果超过 400ms 未被调用，则 Loomo 将停止移动。因此，如果调用频率略大于 400ms, Loomo 就会抖动。

2.不要将 DTS 中检测到的目标距离和方向直接传递给 updateTarget()方法，这会导致 Loomo 撞到目标。正确的方法是从总距离中减去 Loomo 和目标之间的距离。

在某些特殊情况下，会添加一个辅助轮来增强 Loomo 的稳定性。在调用 setCartMode(true)后，Loomo 将停止保持自平衡，但仍然接受速度命令。

```
// stop self-balancing
mBase.setCartMode(true);
mBase.setLinearVelocity(1.0f);
mBase.setAngularVelocity(0.15f);

// recover self-balancing
mBase.setCartMode(false);
```

视觉定位系统(Visual Localization System)

视觉定位系统比里程计更适合用于导航。VLS 基于深度图像计算坐标数据，比起里程计具有更高的精度。

```
// set base control mode
mBase.setControlMode(Base.CONTROL_MODE_NAVIGATION);
// start VLS
mBase.startVLS(true, true, new StartVLSListener() {
        @Override
        public void onOpened() {
                // set navigation data source
                mBase.setNavigationDataSource(Base.NAVIGATION_SOURCE_TYPE_VLS);
                mBase.cleanOriginalPoint();
                PoseVLS poseVLS = mBase.getVLSPose(-1);
                mBase.setOriginalPoint(poseVLS);
                mBase.addCheckPoint(1f, 0);
                mBase.addCheckPoint(1f, 1f);
                mBase.addCheckPoint(0f, 1f);
                mBase.addCheckPoint(0, 0);
        }

        @Override
        public void onError(String errorMessage) {
                Log.d(TAG, "onError() called with: errorMessage = [" + errorMessage + "]");
        }
});
```

注意：

1. 当 VLS 用于导航时，Base 控制模式应设置为 Base.CONTROL MODE NAVIGATION。
2. 导航数据源应该设置为 Base.NAVIGATION_SOURCE_TYPE_VLS, 否则 VLS 无法工作。

六、 Sensor 模块 (Detailed SDK DOC)

传感器 SDK 是一组 API，可用于从 Loomo 的许多传感器中获取数据。这些内置传感器可以检测事件或数值变化。
超声波传感器用于检测障碍物和避免碰撞。超声波传感器安装在 Loomo 的前部，探测距离为[250mm,1500mm]，光束角度为 40 度。
Loomo 还有两个红外传感器，可以观察路况的变化以便顺利行驶。

在使用 SDK 前，需要先初始化 Sensor 实例并绑定到服务:

```
mSensor = Sensor.getInstance();
mSensor.bindService(getApplicationContext(), new ServiceBinder.BindStateListener() {
        @Override
        public void onBind() {
             Log.d(TAG, "sensor onBind");
        }
        @Override
        public void onUnbind(String reason) {
        }
    });
```

获取传感器数据:

```
SensorData mInfraredData = mSensor.querySensorData(Arrays.asList(Sensor.INFRARED_BODY)).get(0);
float mInfraredDistanceLeft = mInfraredData.getIntData()[0];
float mInfraredDistanceRight = mInfraredData.getIntData()[1];

SensorData mUltrasonicData = mSensor.querySensorData(Arrays.asList(Sensor.ULTRASONIC_BODY)).get(0);
float mUltrasonicDistance = mUltrasonicData.getIntData()[0];

SensorData mHeadImu = mSensor.querySensorData(Arrays.asList(Sensor.HEAD_WORLD_IMU)).get(0);
float mWorldPitch = mHeadImu.getFloatData()[0];
float mWorldRoll = mHeadImu.getFloatData()[1];
float mWorldYaw = mHeadImu.getFloatData()[2];

SensorData mHeadPitch = mSensor.querySensorData(Arrays.asList(Sensor.HEAD_JOINT_PITCH)).get(0);
float mJointPitch = mHeadPitch.getFloatData()[0];

SensorData mHeadYaw = mSensor.querySensorData(Arrays.asList(Sensor.HEAD_JOINT_YAW)).get(0);
float mJointYaw = mHeadYaw.getFloatData()[0];

SensorData mHeadRoll = mSensor.querySensorData(Arrays.asList(Sensor.HEAD_JOINT_ROLL)).get(0);
float mJointRoll = mHeadRoll.getFloatData()[0];

SensorData mBaseTick = mSensor.querySensorData(Arrays.asList(Sensor.ENCODER)).get(0);
int mBaseTicksL = mBaseTick.getIntData()[0];
int mBaseTicksR = mBaseTick.getIntData()[1];

SensorData mBaseImu = mSensor.querySensorData(Arrays.asList(Sensor.BASE_IMU)).get(0);
float mBasePitch = mBaseImu.getFloatData()[0];
float mBaseRoll = mBaseImu.getFloatData()[1];
float mBaseYaw = mBaseImu.getFloatData()[2];

SensorData mPose2DData = mSensor.querySensorData(Arrays.asList(Sensor.POSE_2D)).get(0);
Pose2D pose2D = mSensor.sensorDataToPose2D(mPose2DData);
float x = pose2D.getX();
float y = pose2D.getY();
float mTheta = pose2D.getTheta();
float mLinearVelocity = pose2D.getLinearVelocity();
float mAngularVelocity = pose2D.getAngularVelocity();

SensorData mWheelSpeed = mSensor.querySensorData(Arrays.asList(Sensor.WHEEL_SPEED)).get(0);
float mWheelSpeedL = mWheelSpeed.getFloatData()[0];
float mWheelSpeedR = mWheelSpeed.getFloatData()[1];
```

注意：

1. 距离的单位是 mm；角的单位是 rad；线速度的单位是 m/s；角的单位是 rad/s；LeftTicks 和

RightTicks 的单位是 Tick，当轮胎充好气后等于一厘米。
2. 有一个已知的问题，当障碍物与超声波传感器之间的距离小于 250mm 时，可能会返回错误的值。
3. 当您调用 robotTotalInfo.getHeadWorldYaw().getangle()来获取 WorldYaw 的值时，它总是返回 0.0f，因为当前版本还不支持。

七、 坐标系转换(Transformation Frames ,TF)

Tf SDK 为开发人员提供 AlgoTfData 类。通过该类可以获得 Loomo 存储的每个坐标系的坐标转换(Rotation 和 Translation)，例如平台相机坐标系、底座坐标系、里程计原点坐标系。

用户必须使用传感器 SDK 中的 getTfData 方法来获取 Tf 数据。在输入中，必须以成对的方式使用 Loomo 的坐标系，首先指定目标坐标系，然后指定源坐标系。这样会给出目标坐标系相对于源坐标系的 TF 数据。在以下代码中，检索平台相机坐标系相对于底座坐标系的 Tf 数据。-1 表示从最近的时间戳中检索数据：

```
AlgoTfData Tf;
Tf = mSensor.getTfData(Sensor.PLATFORM_CAM_FRAME, Sensor.BASE_POSE_FRAME, -1, 100);
```

如前面所述，Tf 数据提供了空间坐标和旋转坐标。空间坐标为 Translation 类型的“t”变量；旋转坐标为 Quaternion 类型的“q”变量。用户能够从 SDK 中提供的 Quaternion 类恢复 pitch、yaw 和 roll 数据：

```
Tf.t.x; // the x-coordinate of the camera frame w.r.t the base frame
Tf.t.y; // the y-coordinate of the camera frame w.r.t the base frame
Tf.t.z; // the z-coordinate of the camera frame w.r.t the base frame
Tf.q; // the quaternion of the camera frame w.r.t the base frame
Tf.q.getYawRad();
Tf.q.getPitchRad();
Tf.q.getRollRad();
```

通过这些信息，用户能够使用 Translation 和 Quaternion 类计算与机器人相关的其他参考坐标系。下面的代码演示了执行旋转或平移转换的计算操作：

```
//initialization
Quaternion rotation;
rotation = new Quaternion(0,0,0,0);
Translation translation;

//giving values
rotation.setEulerRad((float)Math.PI/2,0,0);
translation = new Translation (0,1,0);

//operations
Tf.q.mul(rotation); //quaterion multiplication, returns quaternion
Tf.q.quaternionRotate(translation); //quaternion and translation multiplication, returns translation
```

八、 表情(Emoji)

Emoji SDK 是一个允许 Loomo 调节和玩弄情绪、声音和动作的框架。这使得机器人更加逼真。通过使用 Loomo 的默认表达库，只需几行代码，就可以使 Loomo "look down," "look up," 或 "look around"。

在 Emoji 版本中，提供了 16 种不同的表情。

Emoji SDK 样例代码：

首先，将 EmojiView 放到你的 layout 布局中：

```
<com.segway.robot.sdk.emoji.EmojiView
        android:id="@+id/face"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
</com.segway.robot.sdk.emoji.EmojiView>
```

第二步，初始化 Emoji。HeadControlManager 是 HeadControlHandler 的一个实现类。在这个类中，我们调用 Head SDK，制作一个接口让 Emoji 控制机器人的头部。注意，在动画开始之前，需要将机器人的头部设置为 Emoji 模式。

```
Emoji mEmoji;
HeadControlManager mHandcontrolManager;

mEmoji = Emoji.getInstance();
mEmoji.init(this);
mEmoji.setEmojiView((EmojiView) findViewById(R.id.face));

HeadControlManager mHandcontrolManager = new HeadControlManager(this);
mEmoji.setHeadControlHandler(mHandcontrolManager);
```

最后，制作一个 Emoji：

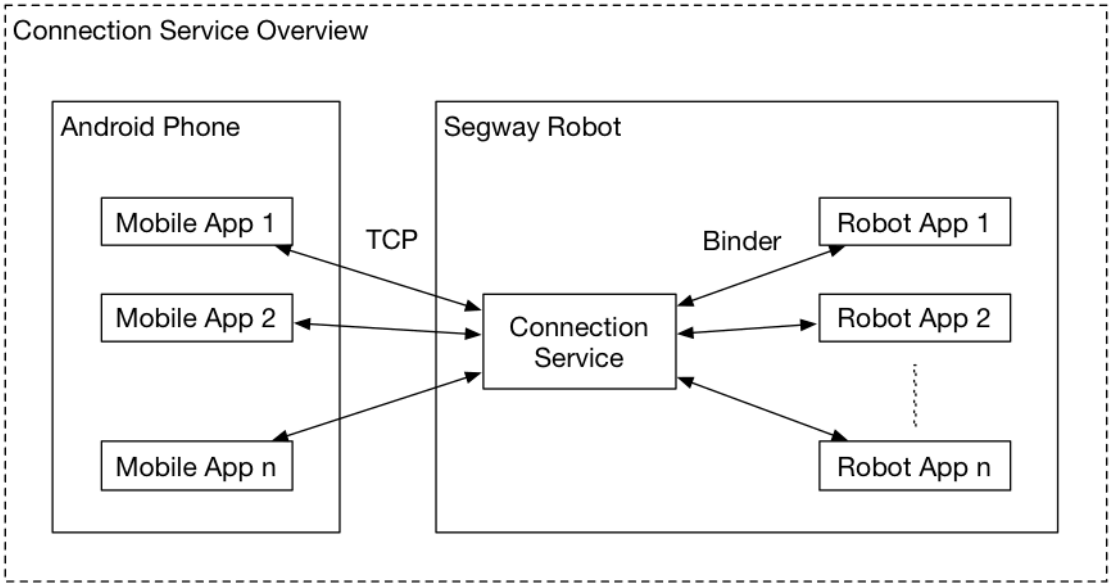
```
mEmoji.startAnimation(RobotAnimatorFactory.getReadyRobotAnimator((Integer) msg.obj), new
EmojiPlayListener() {
   @Override
   public void onAnimationStart(RobotAnimator animator) {
   }

   @Override
   public void onAnimationEnd(RobotAnimator animator) {
      mEmojiView.setClickable(true);
      mHandcontrolManager.setWorldPitch(0.6f);
   }

   @Override
   public void onAnimationCancel(RobotAnimator animator) {
      mEmojiView.setClickable(true);
      mHandcontrolManager.setWorldPitch(0.6f);
   }
});
```

九、 Connectivity 模块 (Detailed SDK DOC For Phone/Robot)

Connectivity 服务帮助开发者在 WiFi 网络下通过手机和 Loomo 交换信息和小体量数据(<1MB)。这项服务可以大大简化使用手机与 Loomo 交互的应用的开发。



Connectivity 在 Loomo 中运行，充当消息处理中心。手机上的应用通过 TCP 连接到该服务，Loomo 上的应用程序通过 Binder 连接到该服务。

在使用该服务前，需要理解一下两个类:

- **MessageRouter 类：**Connectivity 服务最外层的入口类。您可以使用该类的实例连接到服务。
- **MessageConnection 类：**当 Loomo 应用程序和手机都通过 MessageRouter 连接到 Connectivity 服务时，MessageRouter 通过回调函数返回一个 MessageConnection 实例。开发人员可以使用它在手机和 Loomo 应用程序之间进行消息通信。

此外，出于安全考虑，Connectivity 服务使用 AndroidManifest 中的预定义字段来定义连接权限。仅在定义相匹配时执行连接。
例如，如果该手机应用程序的包名为 com.segway.connection.sample，而机器人应用程序上的包名为:com.segway.connection.sample.robot;
手机应用的 AndroidManifest 需要定义以下内容:

```
<meta-data android:name="packageTo1" android:value=" com.segway.connectivity.sample.robot "></meta-data>
```

机器人应用的 AndroidManifest 需要定义以下内容:

```
<meta-data android:name="packageTo1" android:value=" com.segway.connectivity.sample.phone "></meta-data>
```

Connectivity 连接 SDK 分为两部分:机器人端 SDK 和手机端 SDK。这两者需要分别整合。

机器人端样例代码

初始化 SDK 和绑定到 Connectivity 服务：

```
//get RobotMessageRouter
mRobotMessageRouter = RobotMessageRouter.getInstance();
//bind to connectivity service in robot
mRobotMessageRouter.bindService(this, mBindStateListener);
```

如果绑定成功，你需要登录 Connectivity 服务。该服务会检查当前应用的 AndroidManifest 的连接信息，并进行连接匹配：

```
//register MessageConnectionListener in the RobotMessageRouter
mRobotMessageRouter.register(mMessageConnectionListener);
```

在注册时，需要输入一个 MessageConnectionListener 实例，用于在出现新连接时通知开发人员。

当手机端应用程序启动并连接到 Connectivity 服务后，SDK 通过回调函数通知连接已经建立。您可以设置 ConnectionStateListener 来侦听 MessageConnection 的连接和断开，并通过设置 MessageListener 实例来接收消息。

```
@Override
public void onConnectionCreated(final MessageConnection connection) {
    Log.d(TAG, "onConnectionCreated: " + connection.getName());
    mMessageConnection = connection;
    try {
        mMessageConnection.setListeners(mConnectionStateListener, mMessageListener);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

信息可以通过 String 和 byte[]进行发送和接收。

注意：Connectivity 服务被设计用于传输轻量数据,因此传输的数据大小被严格限制，不得超过 1MB。

发送 String：

```
mMessageConnection.sendMessage(new StringMessage("Hello world!"));
```

发送 byte：

```
mMessageConnection.sendMessage(new BufferMessage(bytebuffer.array()));
```

接收消息：

```
private MessageConnection.MessageListener mMessageListener = new
MessageConnection.MessageListener() {
    @Override
    public void onMessageReceived(final Message message) {
        // message received
    }
};
```

手机端样例代码

注意: 在连接之前, 确保 Loomo 和手机在同一局域网上。当前版本需要手机明确知道 Loomo 的 IP 地址。

初始化 SDK，设置 Loomo 的 IP 地址，连接到 Loomo 端的 Connectivity 服务:

```
mMobileMessageRouter = MobileMessageRouter.getInstance();

//robot-sample, you can read the IP from the robot app.
mMobileMessageRouter.setConnectionIp(myRobotIPAddress);

//bind the connectivity service in robot
mMobileMessageRouter.bindService(this, mBindStateListener);
```

对于手机和机器人，检索 MessageRouter 实例、获取 MessageConnection 实例和发送/接收消息的方法是相同的。

附录

一、 样例 APP(Sample App Tutorials)

VISION 模块

VisionSample：GitHub Video
FollowMeSample：GitHub (同时包括 Locomotion 模块的 Base)

SPEECH 模块

SpeechSample：GitHub Video

Locomotion 模块

LocomotionSample：GitHub Video
VLSSample：GitHub
TrackImitator_Robot：GitHub(同时包括 Connectivity 模块)
TrackImitator_Phone：GitHub(同时包括 Connectivity 模块)

Emoji 模块

EmojiVoiceSample：GitHub (同时包括 Locomotion 模块的 Head and Speech 模块)

Connectivity 模块

ConnectivitySample_Robot：GitHub Video
ConnectivitySample_Phone：GitHub

二、 原 Loomo SDK 页面错误集合

1. mBase.(

Make the robot follow somebody under CONTROL MODE FOLLOW TARGET:

```
mBase.setControlMode(Base.CONTROL_MODE_FOLLOW_TARGET);

while(someConditionIsTrue) {
    Person person = getPersonFromDTS();
    if (person.getDistance() > 0.35 && person.getDistance() < 5) {
        mBase.(person.getDistance() - 1.2, person.getTheta());
    }
}
```

应为 mBase.updateTarget(

2. One is able to retrieve the ptich, yaw, and roll data,应为 pitch

As mentioned before, the Tf data provides spatial and rotational coordinates. The spatial coordinates are contained in the "t" variable of the type Translation. The rotational coordinates are contained in the "q" variable of the type Quaternion. One is able to retrieve the ptich, yaw, and roll data from the Quaternion class provided in the SDK.

3. See the detailed Locomotion SDK document.
给的链接有误，原为 Vison 的 SDK 链接，应为 Head 和 Base